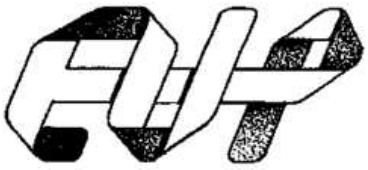
UNIVERSIDAD DE VALLADOLID



ESCUELA UNIVERSITARIA POLITÉCNICA

Implementación en C del Benchmark de Transacciones Distribuidas TPC-C

AUTORES

Eduardo Hernández Perdiguero
Julio A. Hernández Gonzalo

TUTOR

Diego Rafael Llanos Ferraris

Junio 2002

A la memoria de mi abuela,
maestra, consejera y amiga.

A mi familia y a Vanessa.

Agradecimientos

Son muchas las personas a las que tenemos que mostrar nuestro agradecimiento. En primer lugar queremos agradecer a nuestro tutor Diego Rafael Llanos Ferraris su ayuda constante, apoyo y disponibilidad durante el transcurso de todo el proyecto. A nuestras compañeras de laboratorio, Eva y Raquel, por su complicidad, sus consejos y por los buenos momentos pasados. Y por último a nuestras familias y amigos por su apoyo y paciencia a lo largo de toda la carrera.

Prólogo

Este proyecto se enmarca como parte de otro más amplio, que se está desarrollando en el área de investigación del Departamento de Informática de la Universidad de Valladolid. En dicho proyecto se investiga en la posibilidad de construcción de un sistema de computación en paralelo, basado en memoria compartida distribuida, utilizando para ello una red de ordenadores.

El propósito del presente proyecto es doble:

- Por un lado se pretende diseñar un software destinado a medir el rendimiento del sistema anteriormente citado, con el fin de poder hacer una comparativa con otras arquitecturas.
- Por otro se pretende que el realizar un software de libre distribución, con el fin de que esté al alcance de cualquier usuario que desee utilizarlo y/o desarrollarlo.

Se pretende que el software de medición de rendimiento (comúnmente llamado *benchmark*) pueda portarse a cualquier entorno UNIX/LINUX. Con este fin se ha utilizado el ANSI-C como lenguaje base.

No se ha desarrollado un software de medición de rendimientos nuevo, sino que se ha implementado uno siguiendo las especificaciones del benchmark de transacciones distribuidas TPC-C definido por el *Transaction Procesing Performance Council* (TPC) [7]. La versión de TPC-C que se ha implementado es la 5.0 [6]. Al conjunto de programas que forman la implementación se le ha denominado TPCC-UVA.

El TPC-C es un benchmark ampliamente utilizado en la industria para medir el rendimiento de sistemas de procesamiento de transacciones en línea. TPC-C incorpora una mezcla de cinco tipos de transacciones concurrentes de diferente complejidad, para su ejecución en línea o diferida, contra una base de datos en memoria. El TPC-C utiliza como unidad de rendimiento las transacciones por minuto (tpmC). Este parámetro permite comparar la velocidad de servidores en la gestión de transacciones.

Debido a sus características en cuanto a la carga de trabajo que implementa y al forma de procesarla, TPC-C se ha convertido en una buena opción a la hora de medir el rendimiento de sistemas de computación en paralelo.

Si bien las especificaciones de benchmark TPC-C son de dominio público, las diferentes implementaciones sólo están al alcance de de los miembros que conforman el TPC. Los distintos miembros que conforman el TPC utilizan sus implementaciones de benchmark TPC-C, para promocionar sus productos.

Este proyecto viene a cubrir un vacío importante en lo que se refiere a benchmarks de libre distribución, ya que hasta el momento no se había investigado lo suficiente en la realización de un software, de dominio público, basado en el estándar TPC-C.

Índice general

Índice de figuras

Capítulo 1

Introducción

En el presente capítulo pretende que el lector adquiera un conocimiento global sobre el contexto el que se enmarca este proyecto. Para ello en el apartado '1' se exponen una serie de consideraciones acerca de la medición del rendimiento y de la importancia que ello conlleva. En el apartado '2' se pasará a hacer una breve exposición de los tipos de benchmarks para medir el rendimiento. En el apartado '3' se hará una pequeña introducción de los tipos de arquitecturas en paralelo existentes, y un análisis de la importancia que estas están adquiriendo. Finalmente, apartado '4' se presenta el problema que se desea resolver, y se justifica la solución que se ha propuesto para solucionar dicho problema.

1.1. Importancia de las medidas de rendimiento

1.1.1. Introducción

En el presente apartado se hará un análisis de la importancia que el rendimiento tiene en el diseño de una máquina y cómo sirve para determinar las características de ésta una vez diseñada. Además se verá la importancia que el rendimiento tiene para los posibles compradores de un sistema.

La evaluación del rendimiento de un sistema puede ser una tarea compleja debido a la inherente complejidad de los sistemas modernos de programación y a la amplia variedad de técnicas para mejorar el rendimiento empleadas por los diseñadores de circuitos. De hecho, para diferentes tipos de sistemas pueden necesitarse distintas medidas ya que distintos factores del sistema serán los que afectan al rendimiento.

Un motivo importante para calcular el rendimiento es la obtención de la relación precio/rendimiento, que permite hacer una comparativa entre la efectividad

de distintas máquinas. A la hora de distinguir entre diferentes máquinas el rendimiento es siempre un atributo importante. Esto hace que unas medidas precisas y una comparación entre diferentes máquinas sean críticas para los compradores, y por lo tanto, para los diseñadores. Los vendedores de computadores conocen este hecho y a menudo, se esfuerzan por que los compradores conozcan las mejores cualidades de sus máquinas independientemente de si esas cualidades reflejan las necesidades de los compradores. Por lo tanto, el entender cómo medir mejor el rendimiento y sus límites es importante a la hora de seleccionar una máquina.

La realización de la medida del rendimiento va más allá de la simple evaluación desde el exterior de la máquina. Para entender los resultados de rendimiento de una máquina se necesitan entender los factores que lo determinan. Por ejemplo, para mejorar el rendimiento de un sistema, se podría necesitar entender los factores de la circuitería que influyen en el rendimiento global y su importancia relativa. Entender la forma en que estos factores afectan al rendimiento es crucial para entender los fundamentos del diseño de aspectos particulares de la máquina.

1.1.2. Definición de rendimiento

El hecho de decir que una máquina tiene mejor rendimiento que otra puede significar distintas cosas, ya que podemos definir el rendimiento de un computador de varias maneras distintas.

Si se estuviera ejecutando un programa en dos estaciones de trabajo diferentes, se podría decir que la más rápida es la que acaba el trabajo primero. Sin embargo, si se estuviera hablando de un centro de computación que tiene dos grandes computadores de tiempo compartido, los cuales ejecutan trabajos enviados por diferentes usuarios, se diría que el computador más rápido es aquel que ha completado más trabajos durante el día. Esto pone de manifiesto que las métricas adecuadas para especificar el rendimiento dependen de la utilidad que se va a dar al sistema .

Los usuarios individuales de ordenadores, están más interesados en reducir el *tiempo de respuesta*: el tiempo entre el inicio y el final de una tarea, también llamado *tiempo de ejecución o latencia*. Los directivos de los centros de computación están habitualmente interesados en incrementar la productividad (*throughput*): la cantidad total de trabajo hecho en un cierto tiempo, a veces denominado *ancho de banda*. Normalmente los términos *tiempo de respuesta*, *tiempo de ejecución* y *productividad* se utilizan cuando se está desarrollando una tarea de cálculo completa. Los términos *latencia* y *ancho de banda* casi siempre se usan cuando se habla de un sistema de memoria [1].

1.1.3. Productividad frente a tiempo de respuesta

A continuación planteará un ejemplo para plasmar las diferencias entre productividad y tiempo de respuesta. Supongamos que en una máquina se realizan los siguientes cambios:

1. Remplazar el procesador de un ordenador por una versión más rápida.
2. Añadir al procesadores adicionales a un sistema que usa múltiples procesadores para diferentes tareas.

Estos cambios afectan de distinta manera a la productividad y al tiempo de respuesta. En el primer ejemplo se reduce el tiempo de respuesta. Reducir el tiempo de respuesta de un sistema casi siempre mejora la productividad, por lo que en este caso, también la productividad se ve mejorada. En el segundo caso ninguna de las tareas consigue que su trabajo se haga más rápido, por lo tanto sólo la productividad se verá incrementada. Sin embargo, si la demanda de procesadores en el segundo caso fuera casi tan alta como la productividad, el sistema podría verse forzado a poner en cola algunas peticiones, con lo que el incremento de la productividad podría mejorar también el tiempo de respuesta, ya que eso reduciría el tiempo de espera en la cola. Por lo tanto, en bastantes de los sistemas computadores reales, el tiempo de ejecución y la productividad pueden afectarse mutuamente.

1.1.4. Rendimiento y tiempo de ejecución

El rendimiento está directamente relacionado con el tiempo de ejecución . Una forma de aumentar el rendimiento de una máquina es minimizar el tiempo de respuesta o tiempo de ejecución de una tarea. Por lo tanto podemos expresar el rendimiento de una máquina como:

$$Rendimiento = \frac{1}{Tiempo \quad de \quad ejecucion}$$

De esta forma podemos decir que una máquina A tiene más rendimiento que otra máquina B cuando el tiempo de respuesta de A es menor que el de B:

$Rend. \quad A > Rend. \quad B \Rightarrow \frac{1}{T.ejecucion \quad A} > \frac{1}{T.ejecucion \quad B} \Rightarrow$
$\Rightarrow T.ejecucion \quad A > T.ejecucionB$

A la hora de comparar el rendimiento de dos máquinas a menudo se suele usar el *rendimiento relativo*. Decimos que una máquina A es n veces más rápida que una máquina B cuando:

$$\frac{Rend.A}{Rend.B} = n$$

o lo que es lo mismo:

$$\frac{T.\quad ejecucion\quad B}{T.\quad ejecucion\quad A} = n$$

1.1.5. Medición del rendimiento

El tiempo es muy importante en la medición del rendimiento de un ordenador: el ordenador que ejecuta la misma cantidad de trabajo en el menor tiempo es el más rápido. El tiempo de ejecución de un programa se mide en segundos por programa. Pero el tiempo puede ser definido de maneras diferentes, dependiendo de lo que se cuente. La definición más sencilla de tiempo se llama tiempo de reloj *wall clock time*, tiempo de respuesta *responso time* o tiempo transcurrido *elapsed time*. Estos términos se refieren al tiempo total que tarda una tarea en completarse, e incluye los accesos a disco, los accesos a memoria, las actividades de entrada/salida y la carga adicional introducida por el sistema operativo[1].

Sin embargo, a menudo los computadores son de tiempo compartido, y un procesador podría trabajar en diferentes programas simultáneamente. En estos casos, el sistema podría intentar optimizar la productividad más que tratar de minimizar el tiempo de ejecución de un programa concreto. Por lo tanto, a menudo se querrá distinguir entre el tiempo transcurrido y el tiempo que un procesador está trabajando para nosotros.

El *tiempo de ejecución de CPU* o simplemente tiempo de ejecución, es el tiempo que la CPU dedica a ejecutar una tarea concreta y no incluye el tiempo perdido en las actividades de E/S o en la ejecución de otros programas. (nótese, sin embargo, que el tiempo de respuesta que percibe el usuario es el tiempo transcurrido para el programa, no el tiempo de CPU.) Además, el tiempo de CPU puede ser dividido en el tiempo de CPU consumido por el programa, llamado *tiempo de CPU del usuario*, y el tiempo de CPU consumido por el sistema operativo, llamado *tiempo de CPU del sistema*. La distinción entre ambos tiempos es difícil de realizar de una manera precisa, ya que a menudo es complicado el asignar la responsabilidad de las actividades del sistema operativo a un programa de usuario más que a otro.

A veces, se ignora el tiempo de CPU del sistema cuando se examina el tiempo de ejecución global, debido a las imprecisiones cometidas por los sistemas operativos al medir su tiempo de ejecución y a la inexactitud que supone el incluir el tiempo de CPU del sistema cuando se compara el rendimiento de dos máquinas con diferentes sistemas operativos. Por otra parte, el código de sistema en algunas máquinas corresponde a código de usuario en otras, y, como los programas no pueden funcionar sin un sistema operativo que se ejecute por encima de la

circuitería, se puede argumentar a favor de usar la suma de los tiempos de CPU del usuario y del sistema como medida del tiempo de ejecución de un programa.

A menudo, usuarios y diseñadores examinan el rendimiento usando medidas diferentes, ya que para evaluar determinadas características pueden necesitarse métricas de rendimiento específicas. Cabe señalar la dificultad de encontrar una única métrica de rendimiento para evaluar todo tipo de sistemas, y que sea capaz de reflejar las características de todos los componentes de una máquina.

Si se pudieran relacionar las distintas métricas, se podría determinar el efecto de un cambio en el diseño sobre el rendimiento observado por el usuario.

1.1.6. Coste - Rendimiento

A la hora de diseñar un ordenador hay que tener en cuenta, además del rendimiento, el coste. Todos los diseñadores de computadoras deben equilibrar ambos parámetros. Existe un campo de diseño de alto rendimiento, donde éste es el objetivo principal y el coste queda en un segundo plano. La mayoría de las industrias de supercomputación diseñan de esta manera sus productos. En el otro extremo está el diseño de bajo coste, donde este factor tiene un papel primordial respecto al rendimiento. Ordenadores como los clones del IBM PC pertenecen a este grupo. Entre estos dos extremos está el diseño coste/rendimiento, donde los diseñadores equilibran ambos factores. Ejemplos como la industria de estaciones de trabajo muestran con qué tipos de compromisos deben convivir los diseñadores de esta clasificación [1].

En la tarea del diseño de un ordenador se deberá determinar de una forma precisa cómo las distintas alternativas afectan al coste y al rendimiento. La mayoría de los usuarios de ordenadores se preocupan de ambos parámetros. La determinación de cómo las diferentes características de diseño afectan al coste, es a menudo un problema complicado. El coste de una máquina no está únicamente influido por el valor de sus componentes, sino también por los costes del trabajo de ensamblar la máquina, los gastos de investigación y desarrollo, las ventas, la mercadotecnia y por el margen de beneficio. Por estos motivos el diseño de computadores estará siempre regido por el coste y el rendimiento. Encontrar el mejor equilibrio entre ellos será un factor clave en esta tarea.

Por otro lado está la importancia que los compradores dan a la relación precio/rendimiento a la hora de determinar que máquina es la más 'eficiente' y que mejor se ajusta a sus necesidades. Aunque un comprador siempre quiera adquirir la mejor máquina, nunca puede descartar el factor de la economía. El comprador siempre quiere obtener los mejores resultados al mejor precio, por lo que la relación precio/rendimiento se convierte en un factor clave para guiarle hacia la compra de la mejor máquina.

Por lo tanto la relación coste/rendimiento es importante tanto para los diseñadores como para los compradores.

1.1.7. Otras consideraciones acerca del rendimiento

En el diseño de computadores se suele caer en un error habitual, consistente en esperar que la mejora de un solo aspecto de la máquina provoque un aumento del rendimiento proporcional a la mejora de ese aspecto. La política correcta en este asunto la resume la *Ley de Amdahl* o *Ley de los rendimientos descendentes* que dice:

> - *La posible mejora del rendimiento está limitada por la proporción en que se utilice la prestación mejorada* [1].

Un aspecto común en el diseño de circuitería es un corolario de la Ley de Amdahl: hacer rápido el caso más frecuente. La ley de Amdahl nos recuerda que la posible mejora está afectada por el tiempo que consume el suceso. Haciendo más rápido el caso más frecuente se obtiene un mejor rendimiento que optimizando el caso poco común.

Otra consideración acerca del rendimiento es que supone un error considerar que las métricas independientes de la circuitería predicen el rendimiento. Un método, que ha sido utilizado en el pasado, es el uso del tamaño de código como una medida de la velocidad. Con este método, la arquitectura con el programa más pequeño es la más rápida. El tamaño del programa compilado es, por supuesto, importante cuando el espacio de memoria es escaso, pero esto no es lo mismo que rendimiento.

1.1.8. Comparación y resumen del rendimiento

Una vez que se ha seleccionado un conjunto adecuado de programas de prueba y se han obtenido las medidas del rendimiento, se puede escribir un informe sobre las mismas. Al comunicar las medidas de rendimiento se debe seguir el principio de reproductibilidad; se debería hacer una lista que detalle todo lo que la otra persona necesitará para repetir los resultados.

La comparación entre rendimientos no es sencilla, ya que hay que decidir cómo resumir el rendimiento de un conjunto de programas de prueba. Aunque resumir un conjunto de medidas da como resultado una pérdida de información, los vendedores e incluso los usuarios prefieren tener un único número para comparar el rendimiento. El problema es cómo realizar este resumen, ya que esto puede conducir a errores. El siguiente ejemplo ilustra este problema:

Supongamos que el tiempo que tardan en ejecutarse dos programas en dos ordenadores A y B so los siguientes:

	Ordenador A	Ordenador B
Programa 1	1s	10
Programa 2	1000s	100s
Tiempo total	1001s	110s

A partir de los datos anteriores podemos hacer las siguientes afirmaciones:

A es 10 veces más rápida que B para el programa 1.

B es 10 veces más rápida que A para el programa 2.

Tomada individualmente, cada una de estas afirmaciones es cierta. Sin embargo, conjuntamente presentan una idea confusa, por lo que el rendimiento relativo de los computadores A y B no queda claro.

Por lo tanto es muy importante que el resumen del rendimiento se haga de tal forma que no haya posibilidad de equívocos.

1.1.9. Elección de programas para medir el rendimiento

Un usuario de computadores que ejecuta los mismos programas día tras día sería el candidato perfecto para evaluar un ordenador nuevo. El conjunto de programas ejecutados formarían la carga de trabajo. Para evaluar dos sistemas computadores, un usuario simplemente compararía el tiempo de ejecución de la carga en las dos máquinas. Sin embargo, la mayoría de los usuarios no están en esta situación. En su lugar, deben confiar en otros métodos que evalúan el rendimiento de la máquina candidata, esperando que estos métodos reflejen lo bien que la máquina ejecutará la carga de usuario. Esta alternativa normalmente conduce a evaluar la máquina usando un conjunto de programas de prueba (*benchmarks*), los cuales son específicamente escogidos para medir el rendimiento. Los programas de prueba forman una carga con la que el usuario espera predecir el rendimiento de la carga real.

El uso de benchmarks de rendimiento que dependan de una porción de segmento de código muy pequeña anima a hacer optimizaciones a nivel de circuitería o compilador que tengan como objetivo la optimización de estos segmentos. Por ejemplo un compilador podría reconocer trozos de código especiales y generar una secuencia de instrucciones que fuera especialmente eficiente para ese fragmento. De la misma forma, un diseñador podría intentar que cierta secuencia de instrucciones se ejecutara especialmente rápido debido a que pertenece a un programa

de prueba. A menudo, estas optimizaciones pueden ser activadas explícitamente con una opción específica en la compilación, opción que no se usaría cuando se compilen otros programas. Lo que no está claro es si el compilador generará código óptimo o incluso correcto en el caso de que una aplicación real usara estas opciones. A veces, en la búsqueda de un código altamente optimizado para los programas de prueba, los ingenieros introducen optimizaciones erróneas. Los programas pequeños, o los programas que dedican casi todo su tiempo de ejecución a un fragmento de código muy pequeño, son especialmente vulnerables a esos esfuerzos.

Los programas de prueba pequeños suelen ser atractivos en las primeras fases del diseño, ya que son lo suficientemente pequeños para ser fácilmente compilados y simulados, a veces incluso a mano. Son especialmente tentadores cuando los diseñadores están trabajando en una nueva máquina porque los compiladores podrían no estar disponibles hasta una etapa mucho más avanzada del diseño. Los programas pequeños son también más fáciles de estandarizar que los programas más grandes, y de ahí que numerosos resultados sobre rendimiento que han sido publicados estén disponibles para programas de prueba pequeños [1].

A pesar de que podría estar justificado el uso de estos programas de prueba pequeños al principio del proceso de diseño, no hay razones válidas para usarlos en la evaluación de sistemas de computadores que ya estén funcionando. Por ejemplo, en el pasado era difícil obtener aplicaciones grandes que pudieran ser fácilmente llevadas a máquinas distintas, pero eso ya no es cierto en la actualidad.

Hoy en día, se está ampliamente de acuerdo en que una de las mejores formas para la evaluación del rendimiento es el uso de aplicaciones reales. Éstas podrían ser aplicaciones que el usuario emplea regularmente o simplemente aplicaciones típicas. El usar aplicaciones reales como programas de prueba hace mucho más difícil el encontrar formas triviales de acelerar la ejecución de dichos programas. Además cuando se encuentran técnicas para mejorar el rendimiento, éstas tendrán mucha probabilidad de ayuda a otros programas además de los de prueba.

Otra tendencia, muy extendida en la actualidad, es el uso de benchmarks que si bien no son aplicaciones reales, intentan simular aplicaciones de entornos específicos de trabajo. El objetivo de esto es que los datos que proporcionen estos benchmarks sean relevantes para un entorno específico aunque realmente el benchmark no sea una aplicación real. Las consideraciones acerca de este tipo de benchmarks se analizarán en el siguiente apartado.

Cabe destacar que el uso de otra cosa que no sean programas reales o benchmarks que simulen aplicaciones específicas de entorno después de los estudios de diseño iniciales, probablemente lleve a unos resultados engañosos. Por un lado son engañosos para el diseñador ya que le hacen tomar decisiones erróneas. Por otro lo son para el comprador, ya que tendrá una información poco relevante para máquina que está analizando.

1.2. Tipos de Benchmarks

Desde la aparición del primeras máquinas de cálculo automático, el desarrollo tecnológico ha ido acompañado del esfuerzo para poder determinar la calidad y eficacia de los nuevos avances. Como resultado de ese esfuerzo se ha desarrollado enorme variedad de programas de prueba (o *benchmarks*) y de métricas utilizadas para medir el rendimiento de una máquina. Ante esa variedad podemos hacer una división entre benchmarks o métricas de ámbito general: MIPS, MFLOPS, etc, y benchmarks de dominio específico como SPEC, Perfect Club o TPC.

1.2.1. Medidas de rendimiento genéricas

MIPS

El esfuerzo por encontrar una métrica estándar de rendimiento y fácil de usar ha llevado en muchas ocasiones a que métricas simples, válidas en un determinado contexto, sean mal utilizadas en exceso. Un ejemplo de esto son los MIPS (millones de instrucciones por segundo) [2].

El MIPS de un programa se calcula como:

$$MIPS = \frac{Numerodeinstrucciones}{Tiempodeejecucion*10^6}$$

Como MIPS está relacionada con el número de instrucciones ejecutadas, MIPS es inversamente proporcional al tiempo de ejecución, de forma que las máquinas más rápidas tienen un mayor MIPS. La característica positiva de esta métrica es que es fácil de entender y que máquinas más rápidas tienen un MIPS superior, tal y como nos indicaría la intuición.

Existen tres problemas cuando se utilizan MIPS para comparar máquinas. Primero, MIPS está relacionado con el número de instrucciones ejecutadas, pero no tiene en cuenta las características de dichas instrucciones. No podemos comparar ordenadores con diferentes repertorios de instrucciones usando MIPS, ya que el número total de instrucciones puede ser muy distinto. Segundo, MIPS varía entre programas de un mismo computador, de forma que una máquina no puede tener un único MIPS para todos los programas. Tercero y mas importante, MIPS puede variar inversamente al rendimiento [1]. Hay muchos ejemplos que ilustran este resultado. Uno de ellos se relata a continuación.

Supongamos una máquina que tiene un repertorio de tres instrucciones cuyos CPI's (ciclos por instrucción) son:

Instrucción	CPI
A	1
B	2
C	3

Supongamos ahora que medimos el rendimiento de un programa para dos compiladores diferentes que producen la siguiente secuencia de instrucciones:

	A	B	C
Compilador 1	5	1	1
Compilaror 2	10	1	1

Si la frecuencia de reloj de la máquina son 500Mhz, podemos analizar el rendimiento en términos de MIPS y en términos de tiempo de ejecución.

Primero calcularemos el tiempo de ejecución para cada uno de los compiladores de la siguiente forma:

- Tiempo de ejecución $= \frac{Numero \quad de \quad ciclos \quad de \quad CPU}{Frecuencia \quad de \quad reloj}$
- Ciclos de CPU con compilador 1 $= (5 * 1 + 1 * 2 + 1 * 3) * 10^9 = 10 * 10^9$
- Ciclos de CPU con compilador 2 $= (10 * 1 + 1 * 2 + 1 * 3) * 10^9 = 15x10^9$
- Tiempo de ejecución 1 $= \frac{10*10^9}{500*10^9} = 20segundos$
- Tiempo de ejecución 2 $= \frac{15*10^9}{500*10^9} = 30segundos$

Según estos resultados se puede concluir que el compilador 1 genera el programa más rápido de acuerdo con el tiempo de ejecución. Ahora calcularemos el MIPS para cada programa:

- MIPS 1 $= \frac{5+1+1}{20x10^6} = 350$
- MIPS 2 $= \frac{10+1+1}{20x10^6} = 400$

Según esto, hemos llegado a la contradicción de que el compilador 2 tiene un MIPS mayor, mientras que el código generado por el compilador 1 se ejecuta más rápido.

Como demuestra éste ejemplo, MIPS puede fallar al intentar dar una idea real del rendimiento, incluso comparando dos versiones de un mismo programa en la misma máquina.

MFLOPS

Otra de estas métricas simples son los millones de operaciones en punto flotante por segundo, abreviadamente mega FLOPS o MFLOPS, cuya fórmula es:

$$MFLOPS = \frac{N^{o}\,de\,operaciones\,de\,punto\,flotante\,de\,un\,programa}{T.de\,ejecucion\,x10^{6}}$$

Evidentemente, una estimación en MFLOPS depende de la máquina y del programa. Los MFLOPS se pensaron para medir el rendimiento en punto flotante, por lo que no son aplicables fuera de ese rango.

El término MFLOPS está basado en las operaciones en lugar de en las instrucciones, y se pensó para que fuera una comparación buena entre diferentes máquinas. La creencia es que el mismo programa corriendo en computadores diferentes debe ejecutar un número diferente de instrucciones, pero el mismo número de operaciones en punto flotante. Desgraciadamente, los MFLOPS no son fiables, porque el conjunto de operaciones en punto flotante no es consistente con las máquinas. Por ejemplo, mientras una máquina puede no tener instrucción de dividir, otra puede tener instrucción de división, raíz cuadrada, seno y coseno. Otro problema observado es que la estimación en MFLOPS cambia no sólo en la mezcla de operaciones de enteros y punto flotante, sino también en la mezcla de operaciones rápidas y lentas de punto flotante. Por ejemplo, un programa con el 100 % de sumas en punto flotante tendrá una estimación mayor que un programa con el 100 % de divisiones en punto flotante. A veces, para solucionar estos problemas se utilizan los MFLOPS normalizados, que consisten en fijar un número de operaciones en punto flotante en el programa fuente para después dividirlo por el tiempo de ejecución [2].

1.2.2. La necesidad de Benchmarks de dominio específico

No existe ninguna métrica simple que pueda medir el rendimiento de un sistema informático para todas las aplicaciones. El rendimiento de los sistemas varía enormemente de un entorno de aplicación a otro. Cada sistema se diseña típicamente para unos pocos problemas de entorno específicos y puede ser incapaz de realizar otras tareas. Por ejemplo muchas supercomputadoras carecen de software de bases de datos y procesamiento de transacciones, por lo que son inapropiadas para muchas aplicaciones comerciales.

La comunidad científica ha desarrollado benchmarks que miden el rendimiento en cálculos numéricos. Estos benchmarks científicos, por ejemplo, no son apropiados para evaluar el rendimiento de un sistema de base de datos o de procesamiento de transacciones ya que el rendimiento de una base de datos depende fundamentalmente de la velocidad de los algoritmos y no de la velocidad del hardware.

Incluso dentro de la extensa gama de sistemas de base de datos hay una gran diversidad entre el rendimiento de los sistemas en contextos específicos. Un sistema puede ser excelente para transacciones simples de actualización intensiva en una base de datos en línea, pero puede tener un rendimiento muy pobre ejecutando consultas complejas en esa base de datos. Inversamente un sistema que despunte en consultas de toma de decisión puede incluso rechazar el acceso en línea a los mismos datos [3].

Los benchmaks de dominio específico, son una respuesta a la diversidad en el uso de los sistemas informáticos. Los benchmarks de este tipo especifican una carga de trabajo que caracteriza a las aplicaciones típicas de un entorno. El rendimiento obtenido al aplicar esta carga de trabajo en varias máquinas proporciona una estimación del rendimiento relativo en ese entorno.

1.2.3. Instituciones que definen benchmarks estándar

Existen varios consorcios que definen benchmarks estándar de dominio específico, métricas de precio estándar y métodos estándar de medir y presentar esos resultados. Algunos de esos consorcios son:

SPEC (System Performance Evaluation Cooperative) [8]: Un consorcio de vendedores que definen benchmarks de entornos científicos y de estaciones de trabajo.

The Perfect Club [9]: Un consorcio de vendedores y universidades que definen benchmarks para dominios científicos con particular énfasis en las arquitecturas en paralelo o poco comunes.

TPC (Transaction Procesing Performance Council) [7]: Un consorcio de vendedores que definen benchmarks para entornos de bases de datos y de procesamiento de transacciones.

1.2.4. Criterios que definen un benchmark de dominio específico

Para ser útil un benchmark de dominio específico debe cumplir los siguientes criterios [3]:

Relevancia: debe medir el rendimiento de pico y la relación precio/rendimiento de los sistemas cuando se realicen las operaciones tipicas de ese entorno específico.

Portabilidad: debe ser fácil implementar el benchmark en distintos sistemas y arquitecturas.

Escalabilidad: el benchmark debe poderse aplicar tanto en sistemas pequeños como en grandes. Debe ser posible escalar el bechmark para adaptarlo a sistemas mayores.

Simplicidad: el benchmark debe ser comprensible, ya que de otro modo perdería credibilidad.

Como ejemplo de una métrica de rendimiento que no cumple estas condiciones podemos hablar de los MIPS. Los MIPS son en efecto un benchmark sencillo, pero no son una métrica portable, ya que no podemos comparar utilizando los MIPS dos máquinas con diferente repertorio de instrucciones. Los MIPS no son tampoco una métrica escalable, ya que su aplicación a un sistema multiprocesador no es clara. Un sistema multiprocesador compuesto por mil procesadores de 1 MIPS no es equivalente a un único procesador de 1000 MIPS. El principal inconveniente de los MIPS es la irrelevancia, ya que no mide el trabajo útil. Por ejemplo, compiladores diferentes en la misma máquina pueden dar distintas tasas de MIPS.

1.2.5. Los benchmarks SPEC

La Cooperativa de Evaluación del Rendimiento de Sistemas (o SPEC) [8] nació en 1988 y está formada por los representantes de numerosas compañías de computadores. Los fundadores fueron Apollo/Hewlett-Packard, DEC, MIPS y Sun, los cuales llegaron a un acuerdo de ejecutar todos un conjunto de programas y entradas reales. El objetivo era mejorar la medida y la representación del rendimiento de las CPUs, lo cual se logró mediante un proceso de medida más controlado y el uso de programas de prueba más realistas .

La primera edición de los programas SPEC fue el SPEC89, la cual consistía en seis programas de prueba en coma flotante y cuatro programas de prueba enteros. El hecho de tener más programas en punto flotante que enteros hacía que se vieran favorecidas máquinas con un fuerte rendimiento en coma flotante, al usar la media geométrica de los tiempos de respuesta como medida de rendimiento.

Uno de los programas incluidos en el SPEC89 era el 'matrix300' que consistía simplemente en una serie de multiplicaciones de matrices. De hecho, el 99 % del tiempo de ejecución se dedicaba a una sola línea de programa. El hecho de que se dedique tanto tiempo a una sola línea que hace el mismo cálculo muchas veces llevó a varias compañías a desarrollar tecnologías de compilación específicas para mejorar el tiempo de ejecución de este programa, con lo que la pretensión original de utilizar este código para medir rendimientos quedaba desvirtuada.

En 1992 se introdujo un nuevo conjunto de programas de prueba, el SPEC92. Éste introdujo programas de prueba adicionales, suprimió el 'matrix300', y proporcionó medidas separadas para programas en punto flotante y para programas de enteros (SPECfp y SPECint respectivamente).

En 1995 se introdujo el SPEC95, que consta de 10 programas de coma flotante y 8 de aritmética de enteros. Cada grupo se trata de forma separada. Los tiempos de ejecución obtenidos en la máquina a evaluar son en primer lugar normalizados, dividiéndoles por los tiempos de ejecución de una Sun SPARCstation 10/40. Esta normalización produce una medida llamada ratio SPEC que tiene la ventaja que los resultados mayores representan un mayor rendimiento.

SPEC también ha añadido otros grupos de programas de prueba más allá de los grupos originales creados para medir el rendimiento de la CPU. Los SDM (Sistemas de Desarrollo Multimedia) contienen dos programas de prueba que son versiones sintéticas de carga de trabajo de desarrollo (editores, compiladores, ejecuciones, comandos de sistema). El conjunto de programas SFS (Servidor de Ficheros de Sistema) es una carga de trabajo sintética para analizar el rendimiento de un servidor de ficheros. Ambos conjuntos incluyen componentes significativos de E/S y del sistema operativo, no como las pruebas de CPU [1].

1.2.6. Benchmarks de dominio específico para bases de datos y procesamiento de transacciones

Debido a fiabilidad de los benchmarks de entorno específico, se han desarrollado benchmarks para el cálculo del rendimiento de los sistemas de bases de datos y de procesamiento de transacciones. Estos benchmarks se caracterizan por medir el rendimiento del sistema en su totalidad (procesador, sistemas de entrada-salida, sistema operativo, compiladores, gestor de base de datos, etc) en vez de únicamente el rendimiento de CPU. Además incluyen métodos para determinar el precio del sistema (hardware, software, mantenimiento...) a fin de calcular la relación precio/rendimiento.

Durante los últimos veinte años los fabricantes han desarrollado sus propios benchmarks para comparar sus productos con los de la competencia. Raramente los resultados obtenidos se publicaban o se filtraban a la prensa, bien porque los resultados no eran competitivos o porque generalmente eran tratados con escepticismo por parte de los compradores. En ocasiones, si una entidad independiente publicaba resultados para distintas máquinas, los perdedores trataban de desacreditar el benchmark utilizado, lo cual daba inicio a una *guerra de benchmarks* [3]. Una guerra de benchmarks comienza cuando alguien pierde en una evaluación de benchmark reconocida. Los vendedores vuelven a ejecutar los tests utilizando técnicas adaptadas a sus productos consiguiendo tasas de rendimiento ganadoras, entonces, los oponentes vuelven también a ejecutar los tests utilizando sus

propias soluciones y por supuesto consiguen las mejores tasas. Todo esto conduce a una competencia por conseguir los mejores resultados sin variar realmente las características del sistema.

El *benchmarketing* es una variación de las guerras de benchmarks. Para cada sistema hay un benchmark que le considera la mejor opción. El departamento de marketing de la empresa fomenta, entonces, la estandarización de ese benchmark, a menudo sin especificar sus detalles. El *benchmarketing* conduce a la proliferación de benchmarks y crea confusión. Esta práctica ha llevado en los últimos años a un excepticismo universal hacia los resultados de estos bechmarks. La polémica que rodeaba a los benchmaks causó una enorme confusión en la comunidad de desarrolladores en el campo de las bases de datos y de procesamiento de transacciones. En respuesta a esto, un consorcio de 34 vendedores de hardware y software fundó en 1988 el *"Transaction Processing Performace Council"* (TPC) [7]. El objetivo del TPC es definir benchmarks de dominio específico para los sistemas de base de datos y de procesamiento de transacciones, así como proporcionar un método para el cálculo del precio del sistema y la forma en la que los resultados deben ponerse en conocimiento [3].

1.2.7. El consorcio TPC

Los estándares TPC [7] han reducido considerablemente las guerras de benchmark y el *benchmarketing*. Cada vez los clientes confían más en los resultados TPC, por lo que demandan a los vendedores las tasas TPC de sus productos. Las métricas TPC obtienen el rendimiento y el precio/rendimiento de los sistemas de base de datos y de procesamiento de transacciones ejecutando transacciones simples de actualización intensiva. El TPC dispone también de mecanismos que evitan la publicación de resultados que no correspondan con la realidad. El benchmark debe componerse de hardware y software estándar y se debe especificar cualquier sintonización especial o cambio en los parámetros por defecto. Además el TPC recomienda encarecidamente la realización de una auditoría por una entidad independiente.

Algunos estándares definidos por el TPC son el TPC-A [4], el TPC-B [5], y el TPC-C [6]. Describiremos brevemente cada uno de ellos:

TPC-A

El TPC-A pone en juego los componentes del sistema necesarios para realizar tareas propias de los entornos de procesamiento de transacciones en línea (OLTP), haciendo hincapié en las actualizaciones intensivas de base de datos. Este entorno de aplicación se caracteriza por tener:

- Múltiples sesiones de terminales en línea.
- Un número significativo de lecturas y escrituras en disco.
- Tiempos de sistema y de aplicación moderados.
- Integridad de las transacciones.

La métrica utilizada por el TPC-A para expresar el rendimiento son el número de transacciones por segundo (tpsA) y el precio correspondiente por tpsA. Los resultados del TPC-A están sujetos a limitaciones en cuanto al tiempo de respuesta.

El TPC-A es válido para configuraciones de red tanto locales como de área ancha, definiéndose por tanto el tpsA/local y el tpsA/wide. Estas dos métricas son independiente y no pueden compararse entre sí.

La carga de trabajo que realiza este benchmark se compone de una única transacción de actualización intensiva, que intenta representar una aplicación OLTP aunque no simule todas las aplicaciones OLTP posibles.

Todas las especificaciones del benchmark deben recogerse en un Informe Completo de Especificaciones (Full Disclosure Report) que debe estar disponible junto con los resultados obtenidos.

El benchmark simula la carga de trabajo de un banco compuesto por una o más sucursales. Cada sucursal tiene múltiples cajeros. El banco tienen múltiples clientes que poseen una cuenta. En la base de datos se recoge el saldo de cada entidad, cajero y cuenta, y una historia de las últimas transacciones ejecutadas. Las transacciones representan operaciones de imposición y reintegro en una determinada cuenta. Las transacciones se realizan a través de un número determinado de terminales distribuidos por las distintas sucursales, el número de transacciones ejecutadas por segundo determina el rendimiento del sistema.

TPC-B

Al igual que el TPC-A, pone en juego los componentes del sistema para realizar tareas propias de los entornos de procesamiento de transacciones, realizando operaciones de actualización intensiva. El entorno de trabajo se caracteriza por tener:

- Un número significativo de lecturas y escrituras en disco.
- Tiempos de sistema y de aplicación moderados.
- Integridad de las transacciones.

La diferencia con el TPC-A es que este benchmark no simula una aplicación OLTP, por lo que no se requieren terminales, redes ni los llamados *tiempos de pensar*. La unidad de rendimiento de este benchmark es el tpsB o número de transacciones por segundo. El benchmark TPC-B también utiliza una única transacción de actualización intensiva. Todas las especificaciones del benchmark deben recogerse en un Informe Completo de Especificaciones (Full Disclosure Report) que debe estar disponible junto con los resultados obtenidos.

El benchmark simula la misma carga de trabajo que el TPC-A con la diferencia es que las transacciones no se realizan a través de ningún terminal.

TPC-C

En este benchmark también se simula la carga de trabajo característica de los entornos OLTP. Las principales características de este benchmark son:

- Ejecución simultánea de diferentes tipos de transacciones con diferentes grados de complejidad.
- Modos de ejecución de transacciones en línea y en diferido.
- Múltiples sesiones de terminal en línea.
- Un número considerable de accesos a disco.
- Integridad de las transacciones.
- Diseño de base de datos complejo.
- Acceso no uniforme a los datos.

La métrica utilizada es diferente a la de los benchmarks anteriores. En este caso, se utiliza el número de ordenes procesadas por minuto (tpmC).

En este benchmark se definen cinco tipos de transacciones con diferentes características que reflejan la actividad de un ambiente comercial. Un conjunto de terminales solicitan la ejecución de estas transacciones contra una base de datos.

El benchmark TPC-C simula la actividad de un proveedor al por mayor con una serie de distritos distribuidos geográficamente asociados a una serie de almacenes. Cada almacén regional da cobertura a 10 distritos y estos a su vez atienden a 3.000 clientes. Todos los almacenes mantienen un inventario de los 100.000 artículos suministrados por el proveedor.

Estas características hacen que el benchmark TPC-C simule un entorno OLTP de una forma más realista que el TPC-A.

1.3. Introducción a las arquitecturas de computación en paralelo

1.3.1. Importancia de las arquitecturas en paralelo

Aunque las computadoras son cada vez más rápidas, lo que se les exige está creciendo por lo menos al mismo ritmo. La comunidad científica busca continuamente formas de comprobar los límites de sus teorías, y utilizan máquinas de alto rendimiento para simular los sistemas de una forma más realista y con un mayor grado de detalle. Todo esto ha conducido a una creciente demanda de computación de alto rendimiento (HPC) para hacer frente a estos problemas.

Una razón por la que la computación en paralelo se presenta como una de las mejores soluciones es la economía. Haciendo uso de componentes "propios", la computación en paralelo ofrece un rendimiento más alto a unos precios más bajos que máquinas con procesadores diseñados a medida. Además, la inherente escalabilidad de las computadoras en paralelo permite que éstas sean mejoradas, a medida que surgen las necesidades. Mientras que la capacidad de las arquitecturas en serie se mejora dejando obsoleto el procesador anterior, las arquitecturas en paralelo pueden, o al menos en teoría, ser mejoradas simplemente añadiendo más procesadores. Todo esto hace que la computación en paralelo se presente como una de las mejores opciones para acceder a la computación de alto rendimiento a un coste relativamente asequible .

Otra razón que hace interesantes las arquitecturas en paralelo, son las limitaciones físicas que están limitando la velocidad de un único procesador. Por un lado se tiene que la velocidad a la que se puede mover la información dentro del procesador está limitada a la velocidad de la luz, lo cual obliga a diseñar procesadores cada vez más pequeños. Esto conducirá inevitablemente a una situación en la que no se podrá reducir más el tamaño de los transistores por lo que quedara limitado el tamaño mínimo y la potencia de los procesadores. Por otro lado se tienen los problemas de disipación de calor que hacen que en las supercomputadoras se tengan que instalar sofisticados sistemas de refrigeración .

Las razones económicas y las físicas, junto con la inherente escalabilidad de las arquitecturas en paralelo, apuntan hacia un futuro en el que la computación de alto rendimiento estará basada de una u otra forma en las ideas del paralelismo.

1.3.2. Conceptos de la computación en paralelo

En las arquitecturas en serie (arquitectura de Von Neuman), existe un único procesador, que ejecuta en orden una serie de instrucciones y produce unos resultados. Esto es cierto incluso con sistemas operativos que dan la apariencia de

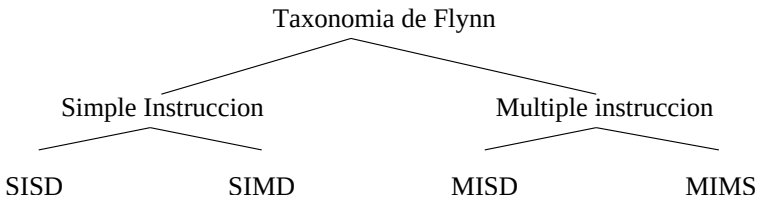


Figura 1.1: Taxonomia de Flynn.

realizar múltiples tareas. En un momento determinado sólo hay una operación realizándose en el procesador [14].

La computación en paralelo se basa en la producción de los mismos resultados usando varios procesadores. El problema a resolver se divide entre los distintos procesadores. Existe una gran variedad de métodos para dividir el problema entre los procesadores disponibles. El dividir el problema de una manera eficiente es fundamental para alcanzar un buen rendimiento en la máquina en paralelo. Lo fundamental en un código eficiente es mantener a todos los procesadores ocupados minimizando la cantidad de comunicaciones, aunque a menudo haya un compromiso entre comunicaciones y procesamiento [11].

1.3.3. Clasificación de las arquitecturas en paralelo

El gran número de arquitecturas en paralelo que han sido propuestas obliga a intentar establecer algún tipo de clasificación. Una clasificación ampliamente utilizada es la que se detalla en la figura 1.1, conocida como *Taxonomía de Flyn* [11].

El gráfico de Flynn se basa en dos conceptos: flujos de datos y flujos de instrucciones. Un flujo de instrucciones corresponde a un contador de programa. Un sistema que tiene 'n' CPUs tiene 'n' contadores de programa, y por tanto 'n' flujos de instrucciones. Un flujo de datos consiste en un conjunto de operandos. Los flujos de instrucciones y de datos son, hasta cierto punto, independientes, por lo que su combinación dan las cuatro posibilidades de gráfico.

Las máquinas SISD ('Simple instrucción, Simple Data') se corresponde con la computadora clásica de Von Newman; tiene un flujo de instrucciones, y un flujo de datos, lo que significa que en cada paso ejecuta una única instrucción sobre un único elemento de datos.

Las computadoras SIMD (un solo flujo de instrucciones, múltiples flujos de datos; 'Single Instruction stream, Múltiple Data stream') se usan para resolver problemas científicos y de ingeniería que son computacionalmente intensivos e implican estructuras de datos regulares como vectores y matrices. Estas máquinas

se caracterizan por tener una sola unidad de control que ejecuta instrucciones una por una, pero cada instrucción opera con varios datos. Las dos especies principales de computadoras SIMD son las matrices de procesadores y los procesadores vectoriales.

La idea en que se basan las matrices de procesadores es que una sola unidad de control proporciona las señales que controlan muchos elementos de procesamiento. Cada elemento de procesamiento consiste en una CPU o ALU ampliada, y normalmente un poco de memoria local. Puesto que una sola unidad de control está conectando a todos los elementos de procesamiento, éstos operan de forma sincronizada [14].

El otro tipo de máquina SIMD es el procesador vectorial. Para explicar el funcionamiento de estas máquinas se propondrá el siguiente ejemplo. Una aplicación de procesamiento de números típica contiene muchos enunciados como

for (i = 0; i <n; i++) a[i] = b[i] + c[i];

donde a, b y c son vectores, casi siempre en punto flotante. El bucle ordena a la computadora sumar los iésimos elementos de b y c y almacenar el resultado en el iésimo elemento de a. Técnicamente, el programa especifica que los elementos deben sumarse en sucesión, pero en este caso el orden no importa. La máquina recibe dos vectores de 'n' elementos y opera con los elementos correspondientes en paralelo; utilizando una ALU vectorial que puede operar con los n elementos simultáneamente y produce como resultado un vector.

Las máquinas MISD son una categoría un tanto extraña, en la que varias instrucciones operan con un mismo dato. No se han construido máquinas de este tipo.

Por último las máquinas MIMD no son más que múltiples CPU independientes que operan como parte de un sistema mayor. Cada procesador puede ejecutar instrucciones individuales, posibilitando a cada uno ejecutar un programa distinto. Casi todos los procesadores paralelos pertenecen a esta categoría. Dentro de esta categoría se puede distinguir entre máquinas de memoria compartida, máquinas de memoria distribuida, y máquinas de memoria compartida virtual [14]. Cada una de estos tipos se detallan a continuación.

1.3.4. Arquitecturas MIMD

Memoria distribuida

En la memoria distribuida cada procesador posee su propia memoria, de forma que cada uno de ellos sólo puede acceder a la memoria a la que está asociado [14]. Cuando un procesador necesita acceder a un dato que está contenido en la

memoria de otro procesador, deberá enviar un mensaje al procesador remoto solicitándole que se le envíe dicho dato. Es por esto que a esta arquitectura también se la suela denominar 'arquitectura de paso de mensajes' o de 'flujo de datos'.

Claramente, el acceso a la memoria local es mucho más rápido que el acceso a la memoria de un procesador remoto. Además cuanto mayor es la distancia física al procesador remoto, más tiempo se tardará en acceder al procesador remoto.

Las arquitecturas de memoria distribuida permiten una mayor escalabilidad que las de memoria compartida, que están limitadas por los cuellos de botella producidos al acceder a ella, como se verá en el siguiente apartado.

Memoria compartida

En la arquitectura de memoria compartida hay un número, normalmente pequeño, de procesadores, cada uno de los cuales accede a una memoria global por medio de algún método de interconexión o bus. Todos los procesadores comparten un único espacio de direcciones. Un procesador se comunica con otro escribiendo los datos en la memoria compartida que luego son leídos por el otro procesador [11].

La ventaja de este tipo de arquitectura es la facilidad de programar, al no existir comunicaciones explícitas entre los procesadores ya que estas se tramitan a través de la memoria compartida con lo que el acceso a ésta es transparente para el programador. Los accesos a la memoria compartida pueden controlarse por medio de técnicas desarrolladas para máquinas multitarea, como por ejemplo semáforos.

No obstante, la memoria compartida tiene limitaciones en cuanto a su escalabilidad. El principal problema ocurre cuando varios procesadores intentan acceder a la memoria compartida al mismo tiempo, produciéndose un cuello de botella. Una forma de evitar estos accesos de memoria conflictivos es dividir la memoria en varios módulos, cada uno conectado al procesador a través de una red de conmutación de alto rendimiento. Sin embargo, este enfoque sólo tiende a mover el problema hacia la red de comunicación.

Existen tres formas de memoria compartida, que se diferencian en la forma de implementarla, que son: Arquitecturas de 'acceso uniforme a memoria' (UMA, Uniform Memory Access), 'acceso no uniforme a la memoria' (NUMA, 'Non Uniform Memory Access), y 'sólo acceso a memoria caché' (COMA, 'Cache Only Memory Access) [14].

Las máquinas UMA (Uniform Accses Memory) tienen la propiedad de que cualquier palabra de memoria se puede leer con la misma rapidez que cualquier otra. Esto es gracias a que todos los procesadores comparten un única memoria

física. Si esto es técnicamente imposible, se frenan las referencias más rápidas de modo que tarden lo mismo que las más lentas y los programadores no noten la diferencia.

Las máquinas NUMA (Non Uniform Memory Access), por el contrario, no poseen la anterior propiedad. En estas máquinas cada CPU tiene un módulo de memoria local lo que hace que el acceso a este sea mucho más rápido que a los distantes. La memoria sigue siendo compartida en un sentido lógico, ya que todos los procesadores siguen compartiendo el mismo espacio de direcciones global, pero es distribuida en un sentido físico. Es por esto por lo que a esta arquitectura se le suela llamar también 'memoria compartida distribuida' (DSM). El hecho de que la memoria esté físicamente distribuida no cancela la transparencia en los accesos a memoria de cara al programador.

Las máquinas COMA (Caché Only Memory Access) utilizan la memoria principal de cada CPU como caché. En este diseño las páginas de memoria no tienen una máquina base fija. En vez de ello, el espacio de direcciones físico se divide en líneas de caché, que se mueven dentro del sistema, según se les necesite.

Memoria Compartida Virtual

A menudo se pueden encontrar máquinas que son una mezcla de las dos arquitecturas anteriores. Un ejemplo de esto es lo que se conoce como arquitectura de Memoria Compartida Virtual. Como en las máquinas de memoria distribuida, cada procesador tiene una memoria local pero el acceso a una memoria remota puede realizarse a través del uso de un espacio de direcciones global. Este acceso remoto es posible gracias a la incorporación de un circuito de soporte que trata con las comunicaciones independientemente del procesador remoto. Esto proporciona unas comunicaciones muy rápidas gracias a uso de un hardware sofisticado (aunque por supuesto no tan rápido como a la memoria local) pero incrementa las çomunicaciones a medida que la distancia que recorre el mensaje se incrementa [11].

1.3.5. Tipos de máquinas en paralelo: Clusters

Por 'Cluster' se entiende un único sistema formado por la unión de distintos computadores, comunicándose entre ellos mediante paso de mensajes, o accediendo todos ellos a un único espacio de direcciones mediante memoria compartida.

A la hora de clasificar una agrupación de máquinas destinadas a trabajar en paralelo, o cluster, podemos distinguirlas en función del grado de unión entre ellas. De esta forma podemos encontrarnos con dos tipos de clusters: Los 'clusters fuertemente acoplados' y los 'cluster débilmente acoplados' [12].

Los clusters fuertemente acoplados son sistemas en los que los distintos procesadores se encuentran cercanos uno a otros, por ejemplo en una misma placa. Estos sistemas suelen utilizar mecanismos de memoria compartida.

Los clusters débilmente acoplados son sistemas en los que los procesadores no está tan cercanos como en el caso anterior. Por ejemplo podrían ser una serie de PC's interconectados entre si. En estos casos se suelen utilizar técnicas de memoria distribuida o memoria compartida distribuida.

1.3.6. Tendencias futuras

En el pasado, una crítica a la computación en paralelo fue la necesidad, y la dificultad técnica, de tener que reescribir el código cuando el usuario deseaba portarlo a una plataforma diferente. La aparición de estándares para computación en paralelo ha hecho frente a ese problema y han animado a usarlos a aquellos usuarios que no estaban dispuestos a perder tiempo y esfuerzo en programar en lenguajes de bajo nivel, pero que quieren aprovecharse de los beneficios de precio/rendimiento que ofrecen las arquitecturas en paralelo.

La posibilidad de desarrollar un programa en, por ejemplo, una estación de trabajo y después llevarlo a una computadora de alto rendimiento, presenta enormes beneficios para los usuarios y hace que la arquitectura en paralelo esté atrayendo a un número cada vez mayor de usuarios.

Los desarrollos tecnológicos y las máquinas que los soportan han evolucionado de una forma significativa, para proporcionar a los usuarios un entorno de programación estable y 'familiar', mediante el desarrollo de herramientas similares a las que el usuario esperaría encontrarse en cualquier máquina serie.

La computación en paralelo se está abriendo a un público cada vez más grande que lentamente se está dando cuenta de las ventajas y el potencial del paralelismo. En sectores tradicionalmente conservadores, tales como el comercial o el financiero, se está adoptando la computación en paralelo. Quizás sectores como éste decidirán el futuro de la computación en paralelo.

1.4. Medición de rendimientos en un cluster. Presentacion del problema

En este proyecto se pretende implementar un benchmark destinado a medir el rendimiento de un cluster experimental que se esta desarrollando en el Departamento de Informática de la Universidad de Valladolid.

El cluster se compone de un conjunto de PC's conectados a través de una red Ethernet y que obedece a la arquitectura de memoria compartida distribuida. Todos los nodos del cluster comparten un único espacio de direcciones pero la memoria se encuentra distribuida físicamente entre ellos. Debido a esto es necesario que los nodos se comuniquen unos con otros mediante paso de mensajes. La tramitación de estos mensajes corre a cargo del sistema operativo, lo que permitirá que está operación sea transparente para el programa en ejecución.

El benchmark que se ha escogido para tal propósito es la versión 5 del benchmark de transacciones distribuidas TPC-C [6]. El nombre adoptado para la implementación ha sido TPCC-UVA.

A continuación se hace un estudio sobre la idoneidad de esta elección.

En la actualidad existen multitud de benchmarks destinados a medir el rendimiento de arquitecturas en paralelo. Estos benchmarks se diferencian en los dominios de aplicación que simulan, así como en sus características de ejecución.

Por un lado están los benchmarks de cálculo numérico puro. Un ejemplo de esto son los *Nas Parallel Benchmarks* [10] que consisten en una serie de programas de cálculo que intentan reflejar aspectos encontrados frecuentemente en aplicaciones de calculo en paralelo destinadas al campo de la aerofísica, como por ejemplo la resolución de ecuaciones de FFT en tres dimensiones.

Este tipo de benchmarks solamente proporcionan información relevante de una parte de la arquitectura en paralelo. En este caso los resultados dan mucha información en cuanto a rendimientos de CPU, pero no reflejan otros aspectos importantes de la arquitectura como pueden ser accesos a discos duros, dispositivos de entrada/salida, etc.

Una de las posibilidades para medir el rendimiento del sistema de una forma global son los benchmarks de bases de datos, ya que presentan una carga de trabajo que incluye los aspectos mencionados anteriormente.

Otro motivo que hace interesantes a los benchmarks de bases de datos es que la información que proporcionan resulta muy relevante en los sectores comerciales. Sectores como el comercial o el financiero se presentan como campos en los que las tecnologías de computación en paralelo juegan y jugarán un papel muy importante. Esto hace que las compañías prefieran este tipo de benchmarks a la hora de decidir qué maquina es mejor para sus propósitos. El motivo principal es que estos benchmarks presentan una carga de trabajo que simula un entorno de aplicación muy similar al de esas empresas.

Entre los benchmarks de bases de datos destacan los patrocinados por el 'Transaction Procesing Performance Council' (TPC) [7]. Algunos de los benchmarks de TPC se centran en actividades de procesamiento de transacciones en línea (On-Line Transacction Processing, OLTP) entre los cuales se encuentra el benchmark

que se ha implementado en este proyecto: el TPC-C.

El TPC-C se basa en la ejecución de una mezcla de transacciones interactivas, y simula características propias de un entorno OLTP tales como encolado o cancelación (rollback) de transacciones. La medida de rendimiento se expresa en transacciones por minuto (tpmC). Este benchmak incluye unos criterios de escalamiento que hacen que el problema sea mucho más realista: el tamaño de la base de datos y el numero de terminales que ejecutan transacciones se ajustan a la capacidad de la maquina a medir, lo que da lugar a una mayor o menor tasa de tpmC.

El TPC-C deriva de otro benchmark OLTP de TPC, el TPC-A, ya descrito en la sección 1.2.7. El TPC-C se diseño con la intención de que fuera más realista que su predecesor, pero manteniendo muchas de sus características. TPC-C requiere de una serie de emuladores de terminal remoto que simulan una población de usuarios con sus terminales que ejecutan transacciones contra una base de datos. Este benchmark simula la actividad de un proveedor mayorista con una serie de distritos de ventas y almacenes distribuidos geográficamente. La carga de trabajo que implementa el TPC-C consiste en ordenes de cliente, ejecución de pagos, ejecución de consultas, repartos, y chequeos de inventario, lo que da lugar a los cinco tipos de transacciones que se implementan en el benchmark.

Comparándolo con el TPC-A, el TPC-C tiene una estructura de base de datos más compleja y múltiples tipos de transacciones con distintos grados de complejidad. Otras características del TPC-C son los requisitos realistas para las pantallas de E/S de los terminales, los accesos a la base de datos a través de llaves primarias así como de llaves no primarias, la completa trasparencia en las particiones de datos, o la cancelación (rollback) de transacciones [13].

Por todos estos motivos es por lo que se ha elegido el TPC-C como el benchmark idóneo para la medición del rendimiento del cluster anteriormente citado.

Las implementaciones existentes del TPC-C no están disponibles a la generalidad de los usuarios, sino que las entidades que forman el organismo TPC poseen sus propias implementaciones para promocionar sus productos. Este proyecto pretende que el benchmark desarrollado se distribuya libremente a fin de que cualquier usuario pueda realizar tests de rendimiento sobre sus sistemas, siendo posible hacer comparativas con otras arquitecturas. El benchmark TPCC-UVA está diseñado para facilitar la portabilidad a cualquier entorno UNIX/Linux compatible con UNIX System V.

1.5. Conclusiones

La eficiencia de un sistema no debe medirse únicamente en términos de rendimiento de CPU, sino que deben ser tomados en cuenta otros aspectos tales como los accesos a disco, las entradas y salidas de datos y la transmisión de información entre las distintas partes de las que se compone el sistema.

Los benchmarks de bases de datos, puesto que ponen en juego todos los recursos mencionados anteriormente, suponen una buena elección para determinar el rendimiento global de un sistema. Por este motivo el benchmark desarrollado, sigue las directrices del benchmark para bases de datos TPC-C.

El resto de la memoria se organiza como sigue. En el capítulo 2 se recogen los detalles sobre la implementación de la primera versión del benchmark TPCC-UVA. En el capitulo 3 se describen los pasos necesarios para la instalación del benchmark y la obtención de resultados. En el capítulo 4 se explica en profundidad características avanzadas del benchmark destinadas a aquellos programadores que quieran desarrollar versiones posteriores de TPCC-UVA. En el capítulo 5 se incluyen algunos resultados obtenidos por este benchmark, además se realiza un análisis de cada uno de ellos. Finalmente en el capítulo 6 se exponen las conclusiones de este proyecto y las líneas de trabajo futuro.

Además en el documento se adjuntan los siguientes apéndices. En el apéndice A se incluye una descripción detallada del estándar TPC-C. El apéndice B es una introducción al modelo relacional de base de datos y al lenguaje SQL. En el apéndice C se describen algunas herramientas que proporciona el motor de base de datos postgreSQL-7.1.3 utilizado en este benchmark. Finalmente en el apéndice D se detallan los aspectos del programa Gnuplot que se utilizan para la obtención de las gráficas que se detallan en las especificaciones del estándar TPC-C.

Capítulo 2

Desarrollo e implementación del benchmark TPCC-UVA

Este capítulo recoge los aspectos relacionados con el desarrollo e implementación del benchmark TPCC-UVA basado en el estándar del Benchmark de Transacciones Distribuidas TPC-C [6] del *Trasactional Processing Performance Council* (TPC) [7]. En el apartado 1 se explicarán las h rramientas de trabajo que se han usado en el proyecto. En el apartado 2 se explicará en qué consiste el benchmark implementado para dar una idea general de su funcionamiento. Finalmente en el apartado 3 se profundizará en los aspectos relacionados con la implementación del benchmark.

En las secciones de este capítulo se harán continuas referencias las cláusulas del estándar TPC-C que se incluyen en el apéndice A.

2.1. Entorno y herramientas del proyecto

En este apartado se describen las herramientas y los métodos de diseño que se han utilizado para la realización del proyecto.

La filosofía del proyecto ha sido crear un benchmark de libre distribución que pudiera ejecutarse en cualquier entorno UNIX/Linux. Con este fin, para su desarrollo se han utilizado exclusivamente programas de libre distribución. Las principales herramientas que se han utilizado son:

- El sistema operativo utilizado ha sido la distribución de Linux Red Hat 7.0 [21, 22].
- Para la compilación de los programas se ha utilizado la GNU Compiler Colection (gcc). Además el código del software se ha escrito en lenguaje

ANSI-C, realizando las operaciones con la base de datos en lenguaje SQL embebido en C.

- El servidor de bases de datos que utiliza el benchmark es postgreSQL 7.1.3 [24].
- Para mostrar de las gráficas de rendimiento que se especifican en el estándar TPC-C se utiliza el visualizador de gráficas Gnuplot [29].

A continuación se pasa a describir detalladamente estas herramientas.

2.1.1. Sistema Operativo Linux

Linux es una versión de UNIX de libre distribución. Su desarrollo comenzó en la Universidad de Helsinki, en Finlandia, a manos de un estudiante llamado Linus Torvalds. Tras esto Linux continuó siendo desarrollado por multitud de programadores de todo el mundo, que gracias a Internet pudieron ir coordinando el proyecto.

Cabe destacar que a pesar de ser en un 95 % UNIX, el kernel de Linux no usa ni siquiera una línea de código del UNIX de la empresa AT&T, dueña actual de los derechos de UNIX, ni de ningún otro UNIX comercial.

Buena parte del software del SO Linux esta bajo licencia GNU. Linux Torvalds se inspiró en otro sistema operativo desarrollado por Andrew Tenenbaum, el MINIX, un SO pensado con fines educativos. La meta de Linux era mejorar este SO que presentaba sus limitaciones por ser un sistema pensado para enseñar, no para fines comerciales.

La versión original del Kernel, la 0.0.1, ni siquiera funcionó y se tuvo que esperar hasta el 5 de Octubre de 1991 cuando Linux anunció la versión 0.0.2 donde ya se podía ejecutar el shell GNU bash y el gcc (compilador de C de GNU). Poco después de la aparición de la versión 0.0.3, apareció la 0.1.0 gracias a la ayuda de una comunidad de programadores que ayudaron en el proyecto. Por fin, y tras muchas revisiones, en el año 1992 se logro la primera versión libre de errores, la 1.0. Actualmente la última versión estable del Kernel es la 2.4.18.

Hoy en día Linux se perfila como uno de los mejores SO que existen, compitiendo con SO's líderes en el mercado.

Actualmente existen empresas que están distribuyendo Linux con ofertas diferentes en función de lo que se quiera realizar. Empresas como Caldera, Red Hat Linux y el grupo de desarrollo Debian, son lideres en este mercado, cubriendo con sus distribuciones la mayoría de las necesidades, tanto de las empresas como de los usuarios domésticos.

A continuación se detallan los componentes que se suelen incluir en las distintas distribuciones disponibles:

Boot Manager: El boot manager o administrador de inicio es una herramienta que reside en el sector de arranque del disco duro y que permite seleccionar entre el sistema Linux y otro que ya estuviera anteriormente en el sistema, como por ejemplo Windows. Existen varios administradores de inicio en el mercado como el System Comander de V Communication o el Boot Manager, pero Linux viene con su propio administrador de inicio, el LILO o Linux Loader y que se instala por omisión. Actualmente existe otro administrador de arranque para linux llamado GRUB.

Sistema X Window: X Window es el subsistema gráfico que soporta la interfaz de usuario gráfica. Uno de los servidores X más comunes es el Xfree86. También existe un modelo comercial llamado simplemente X.

Interfaz de usuario: La línea de comando es la interfaz por omisión de Linux, pero existen entornos gráficos como el KDE (Kool Desktop Environment), el CDE (Commond Desktop Environment) y el GNOME (GNU Network Object Model Environment).

Servicio de Internet: Linux tiene soporte de TCP/IP desde sus comienzos y además de todos los protocolos Internet comunes como DNS (Domain Name Service), http (Hiper Text Transfer Protocol), SMTP (Simple mail Transfer Protocol, correo electrónico), NNTP (para foros de noticias) y PPP/SLIP para enlaces serie como el dial-up. La mayoría de las distribuciones más famosas incluyen el servidor de pagina Web más usado en Internet, el Apache, que es un servidor de código abierto. Así también incluyen gran variedad de clientes de correo, visualizadores de paginas web y de foros de noticias.

Servicio de Impresión de Archivos: Estos servicios permiten al sistema acceder a recursos de red y compartir archivos e impresoras con otros usuarios. Para compartir archivos con otros sistemas UNIX se encuentra el NFS (Network File System). Para el intercambio de archivos con WINDOWS se encuentra el SAMBA que hace que Linux pueda utilizar recursos de una red Windows NT.

Aplicaciones: Actualmente gran cantidad de empresas de software están escribiendo programas para Linux, como Netscape, Oracle, Sun Soft y muchas otros. Pero hay un largo camino por seguir hasta que los fabricantes de hardware abran totalmente las puertas de sus arquitecturas lo que facilitará que se gestione software más útil y con mayor calidad.

Administración de Paquetes: Las distribuciones pueden venir de 3 formas distintas. En paquetes RPM (Red Hat Package Manager), DEB (Para el administrador de paquetes de Debian) y TGZ o Tarballs. Los RPM fueron creados como una forma de distribuir aplicaciones por la empresa Red Hat y

es una de las más usadas. Los paquetes DEB fueron creados por el grupo que lidera la empresa Debian y es otra forma muy eficiente de administración de paquetes. Finalmente los TGZ son la forma primitiva como se distribuía Linux. Existen todavía distribuciones muy usadas, como la Slackware, que todavía utilizan este medio.

Herramientas para el programador: Las distribuciones incluyen un sin fin de componentes y herramientas para el desarrollo, además de lenguajes completos como el C, Pyton, Perl y TCL. Además se incluye el compilador de C de licencia GNU 'gcc'.

2.1.2. El compilador GCC

El benchamark TPCC-UVA ha sido programado en ANSI-C. Para la compilación de los programas se ha utilizado el compilador GCC [23].

GCC es una colección de compiladores que admite varios lenguajes: C, C++, Objective C, Chill, Fortran y Java, así como también las librerías para éstos. Surgió de la imposibilidad de compilar en un entorno que no fuese PC bajo MS-DOS.

Las siglas GCC significaban GNU C Compiler (Compilador C GNU). En la actualidad, al admitir una colección de compiladores, la siglas han pasado a significar GNU Compiler Collection (Colección de compiladores GNU).

El compilador se distribuye bajo la licencia GPL (General Public License) lo que lo hace de libre distribución: se pueden hacer copias de él siempre que se incluya el código fuente (o se indique cómo conseguirlo) y se mantenga la licencia.

Existen versiones para prácticamente todos los sistemas operativos. Viene incluido en la mayoría (si no en todas) las distribuciones de GNU/Linux. La versión DOS de este compilador es el DJGPP.

GCC suministra al usuario muchas herramientas de comprobación de errores, integra una eficiente herramienta de depuración y dispone de muchas opciones de optimización de código, basándose en el microprocesador de destino u optimizaciones sobre la compilación de código inteligente.

Otras características importantes a resaltar son:

- Integración del compilador de Java GCJ.
- Eliminación del código muerto utilizando la representación SSA.
- Preprocesador C integrado en los compiladores C, C++ y Objective C.

- Potentes optimizaciones en las llamadas a subrutinas optimizando los accesos al STACK (pila del procesador)

2.1.3. El modelo relacional y el lenguaje de bases de datos SQL

La base del datos que se utiliza en el benchmark TPCC-UVA está basada en el modelo relacional. Toda la nomenclatura, así como el propio diseño de la base de datos está basado en este modelo.

El modelo relacional es un enfoque que en la actualidad se usa en muchos de los sistemas de bases de datos , y en concreto es el que ha utilizado en este proyecto. Se puede decir que el modelo relacional es una forma de ver los datos o un método de ver los datos y manipularlos. Este modelo se basa en un único concepto para representar los datos: la *relación*, una tabla bidimensional donde se almacenan los datos [19].

El lenguaje que se ha utilizado para implementar las operaciones con la base de datos en el SQL ('Structured Query Language' o Lenguaje de Consulta Estructurado), que está basado en el modelo de bases de datos relacionales. Los sistemas de bases de datos relacionales más comunes consultan y modifican la base de datos mediante este lenguaje. Buena parte del SQL se basa en el álgebra relacional, aunque ofrece muchas características importantes que van más allá de lo ésta ofrece: por ejemplo, la agregación (sumas y conteos, entre otras cosas) y las actualizaciones de la base de datos.

El lenguaje SQL consta de muchos dialectos, existiendo dos principales: el SQL ANSI ('American National Standards Institute') y un estándar actualizado que se adoptó en 1992, llamado SQL-92 o SQL2 [19].

En este proyecto se utiliza el lenguaje SQL embebido en C, es decir, se insertan sentencias de SQL en el código escrito en C. Para poder después compilar el código con gcc, es necesario *preprocesarlo* con un preprocesador de SQL, que se encargue de convertir las sentencias en funciones de C.

En el apéndice B se incluye una descripción de los principios en los que se basa el modelo relacional, así como de las características del lenguaje SQL que se han utilizado para la implementación de benchmark TPCC-UVA.

En el apéndice C de habla del preprocesador de SQL que se ha utilizado en el proyecto: el *ecpg* [26, 27].

2.1.4. Motor de bases de datos postgreSQL

PostgreSQL es un sistema gestor de bases de datos objeto-relacional (ORDBMS) basado en el paquete postgres escrito en la Universidad de California en Berkeley. El proyecto postgres encabezado por el profesor Michael Stonebraker, estuvo patrocinado por diversos organismos oficiales de EE.UU como la agencia de proyectos de investigación avanzada para la defensa DARPA, la oficina de investigación de la armada ARO y la fundación nacional para la ciencia y la empresa ESL.Inc. PostgreSQL es un software de libre ditribución que proporciona soporte para los lenguajes SQL92/99 y otras características más actuales.

Tras una década de desarrollo, postgreSQL se ha convertido en el gestor de bases de datos de libre distribución más avanzado de hoy en día, ofreciendo control de concurrecia multi-versión y soportando casi todas las sintaxis SQL (incluyendo subconsultas, transacciones, y tipos y funciones definidas por el usuario). También cuenta con un amplio conjunto de enlaces con lenguajes de programación incluyendo C, C++, java, perl, tcl y pyton.

Introdujo muchos conceptos de las bases de datos objeto-relacionales que actualmente están disponibles en muchas bases de datos. Los gestores de bases de datos relacionales tradicionales emplean un modelo consistente en una colección de relaciones o tablas identificadas mediante un nombre que contienen atributos de un tipo de datos determinado. En los sistemas actuales los tipos de datos incluyen: números en coma flotante, enteros, cadenas de caracteres, cantidades monetarias y fechas. Es un hecho reconocido que este modelo es inapropiado para las aplicaciones de base de datos futuras. postgreSQL incorpora mecanismos que aumentan su potencia para manejar bases de datos de tal modo que los usuarios pueden adaptarlo fácilmente a su sistema. Estos mecanismos son:

- El soporte para:
  - La inherencia.
  - La definición de tipos de datos.
  - La definición de funciones.
- Restricciones.
- Disparadores.
- Reglas.
- Integridad de transacciones.

Estas características sitúan a postgreSQL dentro de la categoría de las bases de datos Objeto-Relacionales. Hay que recalcar que estas bases de datos no son

distintas que las Orientadas a Objetos que generalmente no soportan bien los lenguajes de bases de datos relacionales tradicionales. PosgreSQL tiene algunas características que son propias del mundo de las bases de datos Orientadas a Objetos, de hecho algunas bases de datos comerciales han incorporado recientemente características en las que postgreSQL fue pionero [25].

A parte del uso que en este proyecto se da a postgeSQL como motor de base de datos, éste ofrece dos herramientas que se han utilizado en la implementación del benchmark TPCC-UVA: el preprocesador de SQL embebido *ecpg* [26, 27], y el Monitor Interactivo *psql* [27]. EL el apéndice C se incluye una explicación de estas dos herramientas.

2.1.5. Visualizador de gráficas Gnuplot

En este proyecto, se utiliza el programa Gnuplot [29] para la visualización de gráficas de resultados que se especifican en la cláusula 5.6 del estándar TPC-C, a partir de los puntos que calcula el TPCC-UVA para dichas gráficas.

Gnuplot es un programa interactivo de trazado de gráficas que permite introducir ordenes en su línea de comandos. Puede utilizarse para trazar funciones y puntos de datos ,tanto en dos como en tres dimensiones en múltiples formatos. Se adapta a muchas de las necesidades que hoy en día tienen los científicos a la hora de representar gráficas. Aunque Gnuplot posee copyright es de libre distribución.

Gnuplot ofrece:

- Trazado en dos dimensiones de gráficas y puntos de datos en diferentes formatos: líneas, puntos, barras ...
- Cálculos en aricmética entera, flotante y compleja.
- Visualización en tres dimensiones con distintos estilos de superficie.
- Definición de funciones de usuario.
- Soporte para un gran número de Sistemas Operativos, formatos graficos y dispositivos.
- Ayuda en línea.
- Etiquetas para títulos, ejes y puntos.
- Línea de comandos con capacidades de edición y memoria.

Las funcionalidades ofrecidas por Gnuplot que se han utilizado para la implamentación de este proyecto se detallan en el apéndice D.

2.2. Consideraciones de Diseño

El benchmark TPCC-UVA ejecuta un test de rendimiento cuya carga de trabajo viene determinada por la actividad de un conjunto de terminales que solicitan, a un sistema de base de datos, la ejecución de una mezcla de cinco tipos de transacciones de actualización y lectura intensiva, tratando de simular, por tanto, la actividad representativa de un entorno OLTP (Online Transaction Processing).

La carga de trabajo simula la actividad de un proveedor al por mayor con distritos de venta y almacenes distribuidos geográficamente. Cada almacén suministra a 10 distrito y cada distrito da servicio a 3.000 clientes. Todos los almacenes mantienen inventarios para 100.000 artículos. Sus características se describen en la cláusula 1 del estándar TPC-C.

Los cinco tipos de transacción que constituyen la carga de trabajo son:

New-Order Introduce una orden completa a través de una única transacción de base de datos.

Payment Realiza el pago de un cliente, actualizando su balance actual.

Order-Status Comprueba el estado de la ultima orden un cliente.

Delivery Registra el reparto de una orden pendiente para cada distrito de un almacén.

Stock-Level Realiza un recuento de los artículos cuyas existencias son menores que un umbral dado.

En todas las transacciones salvo la transacción Delivery, los terminales envían la solicitud de transacción y esperan a la respuesta de ejecución. A esto se le denomina ejecución interactiva. En la transacción Delivery los terminales únicamente generan los datos de transacción y no esperan a la respuesta, ya que esta transacción tiene un modo de ejecución aplazado (ver perfiles de transacción de la cláusula 2 del estándar TPC-C).

Los cinco tipos de transacciones se ejecuta según una mezcla ponderada con los siguientes porcentajes:

- 43,47 % de transacciones New-Order.
- 43,47 % de transacciones Payment.
- 4,35 % de transacciones Stock-Level.
- 4,35 % de transacciones Delivery.

- 4,35 % de transacciones Order-Status.

Mediante estos porcentajes se cumplen los requisitos del estándar TPC-C relativos a la mezcla de transacciones ejecutadas (ver cláusula 5.2.3 del estándar).

El rendimiento ofrecido por el sistema se expresa en transacciones de tipo New-Order por minuto (tpmC), que se han completado durante el test de rendimiento. Nótese que el número de transacciones de tipo New-Order ejecutadas está limitado por los porcentajes anteriores.

La estructura de la base de datos contra la que se ejecutan las transacciones, se compone de nueve tablas, con distintas características en cuanto a su esquema y cardinalidad. Sus características se describen en la cláusulas 1.2 y 1.3 del estándar TPC-C.

Para la ejecución de las transacciones, y la implementación de la estructura de la base de datos se utiliza el motor de base de datos postgreSQL-7.1.3 [24]. Vários de los módulos que componen en benchmark TPCC-UVA iteractúan con este motor.

El benchmark TPCC-UVA se compone de cinco módulos que interactúan entre si a fin de realizar todas las funciones necesarias para la medición del rendimiento de una máquina UNIX/Linux. Estos cinco módulos son:

- Controlador del Benchmark.
- Conjunto de Emuladores de Terminal Remoto (ETR).
- Monitor de Transacciones (MT).
- Controlador de Checkpoints.
- Controlador de Limpiezas.

El esquema general del benchmark se describe a continuación a partir de la figura 2.1.

2.2.1. Controlador del Benchmark

Proporciona la interfaz con el usuario para el acceso a las funciones del Benchmark. Además, realiza las operaciones necesarias para llevar a cabo esas funciones, que son:

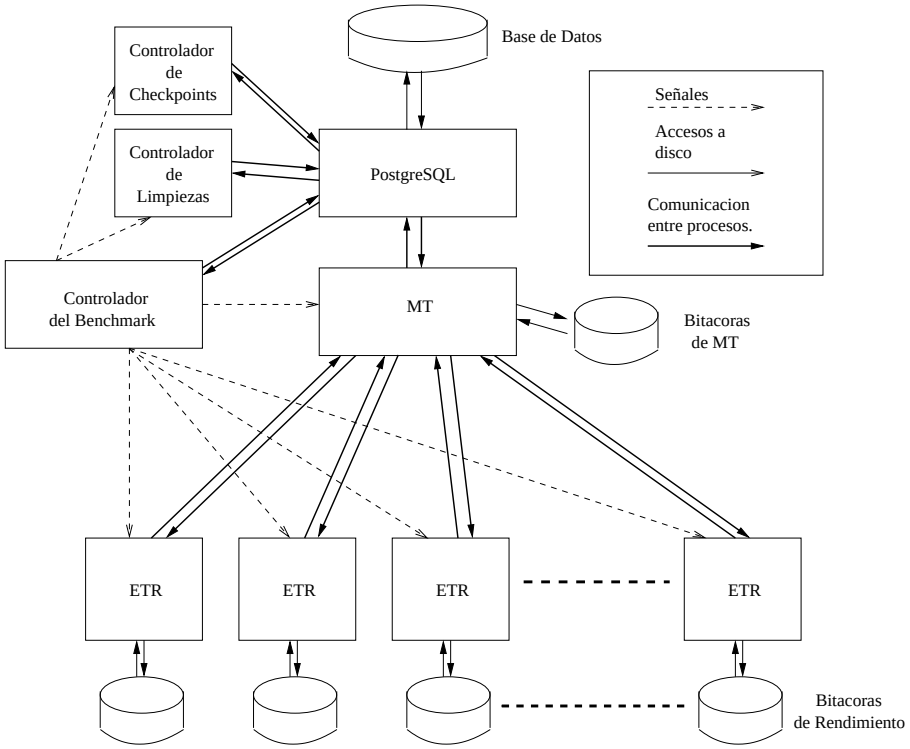


Figura 2.1: Esquema general del benchmark

Población inicial de la base de datos

Se crea una nueva base de datos para el test, compuesta de las nueve tablas definidas en la cláusula 1 del estándar TPC-C y los requisitos de escalamiento y población de la cláusula 4 de dicho estándar, incluyendo, además, mecanismos que aseguran la integridad referencial (primary keys, foreign keys). La base de datos se creará siempre y cuando no exista una previamente.

Restauración de una base de datos existente

Se eliminan las modificaciones producidas por un test en la base de datos. El objetivo de esta función es obtener una base de datos con los requisitos necesarios para realizar un nuevo test, evitando la necesidad de crear una base de datos nueva debido al tiempo que conlleva dicha operación.

Ejecución del test

Se ponen en funcionamiento las partes del Benchmark utilizadas para la ejecución del test de rendimiento, considerando los parámetros seleccionados por el

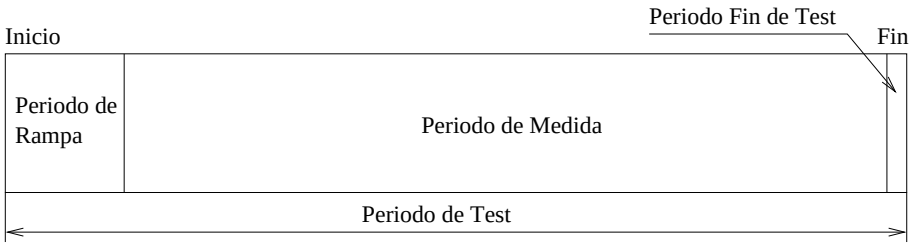


Figura 2.2: Periodo de Test

usuario: número de almacenes, número de terminales por almacén, periodo de rampa, periodo de medida y la configuración del Controlador de Limpiezas.

El estándar TPC-C especifica que los test de rendimiento han de realizarse con 10 Emuladores de Terminal Remoto por cada almacén configurado (cláusula 4.2.2). El TPCC-UVA permite realizar tests con menos ETR.

El controlador del benchmark se encarga de activar el Monitor de Transacciones, el número de Emuladores de Terminal Remoto por cada almacén, seleccionados por el usuario (ver cláusula 4.2.2 del estándar TPC-C), el Controlador de Checkpoints y en su caso el Controlador de Limpiezas. Además se realiza la temporización de los periodos de los que se compone el test, informando a los módulos involucrados en la medición del rendimiento, mediante envío de señales, de la expiración del Periodo de Medida para provocar su desactivación.

El test se compone de dos periodos: Periodo de Rampa y Periodo de Medida. La unión de ambos, junto con el Periodo de Fin de Test se conoce como Periodo de Test. La figura 2.2 representa la disposición de los periodos de los que se compone el test.

Periodo de Rampa: es el periodo en el cual se lanzan los ETR y el MT. Permite que la medida de rendimiento se produzca cuando se haya alcanzado un estado estable como se especifica en la cláusula 5.5.1 del estándar TPC-C.

Periodo de Medida: es el Periodo en el cual se comprueba el rendimiento del sistema.

Periodo de Fin de Test: es el periodo en el que el Controlador del Benchmark provoca el fin del test, desactivando los módulos involucrados.

Comprobación de la consistencia de la base de datos

Se comprueba que la base de datos cumple los requisitos de consistencia mínimas especificadas la cláusula 2.3.3 del estándar TPC-C.

Eliminación de la base de datos existente

Se procede a la eliminación de la base de datos, en el caso de que exista.

Recuento de resultados del ultimo test realizado

Se realiza la lectura y el tratamiento de los ficheros de bitácora producidos por el conjunto de ETRs y el MT que contienen la información necesaria para determinar el rendimiento alcanzado por la máquina. Los resultados que se obtienen de dicho análisis son los siguientes:

- Instante de inicio y de fin del periodo de medida.
- Número de transacciones New-Order ejecutadas por minuto expresado en tpmC.
- Número total de transacciones.
- Para cada tipo de transacción se muestra:
  - Porcentaje de transacciones ejecutadas de cada tipo con respecto al total.
  - Número total de transacciones ejecutadas durante el periodo de medida y durante el periodo de test.
  - Porcentaje de transacciones con un tiempo de respuesta de transacción menor que 5 segundos en el caso de las transacciones New-Order, Payment, Order-Status y Delivery y de 20 segundos en el caso de la Stock-Level, como se especifica en la cláusula 5.2.5.7 del estándar TPC-C . En el caso de la transacción Delivery este tiempo corresponde al tiempo de respuesta de encolado, no el de ejecución.
  - Tiempo de respuesta de transacción mínimo, máximo, medio y para el cual se ejecutan el 90 % de las transacciones.
  - Tiempo de pensar mínimo, máximo y medio, simulados por los ETR (ver cláusula 5.2.5.4 del estándar TPC-C).
  - Para las transacciones New-Order además se muestra:
    - Porcentaje de transacciones canceladas como consecuencia de utilizar un número de artículo no válido, según se especifica en la cláusula 2.4 del estándar TPC-C.
    - Número medio de artículos por orden.
    - Porcentaje de artículos suministrados por un almacén distinto al del terminal de origen (artículos remotos), siempre y cuando exista más de un almacén (ver cláusula 2.4 del estándar TPC-C).

- Para las transacciones Payment además se muestra:
  - Porcentaje de transacciones en las que el pago lo efectúa un cliente de otro almacén diferente al del terminal de origen, siempre y cuando exista más de un almacén (ver cláusula 2.5 del estándar TPC-C).
  - Porcentaje de clientes seleccionados por su identificador de cliente (C_ID) y por su Nombre (C_LAST) (ver cláusula 2.5 del estándar TPC-C)..
- Para las transacciones Order-Status además se muestra:
  - Porcentaje de clientes seleccionados por su identificador de cliente (C_ID) y por su Nombre (C_LAST)(ver cláusula 2.6 del estándar TPC-C)..
- Para las transacciones Delivery además se muestra:
  - Porcentaje y número de almacenes que se han saltado en el reparto (ver cláusula 2.6 del estándar TPC-C).
  - Tiempo de respuesta de ejecución mínimo, medio y máximo.
  - Porcentaje de transacciones que se han ejecutado en menos de 80 segundos.

El usuario puede guardar estos resultados en un fichero.

Tras el análisis, se generan los ficheros con los puntos que permitirán la visualización de las gráficas de resultados que se especifican en la cláusula 5.6 del estándar TPC-C. Para la visualización de estas gráficas se utiliza el programa Gnuplot [29], y son:

- Distribución de los tiempos de respuesta para cada tipo de transacción: representa el tiempo de respuesta de transacción contra el número de transacciones completadas en ese tiempo. Estas gráficas se definen en la cláusula 5.6.1 del TPC-C.
- Distribución del tiempo de pensar para la transacción New-Order: representa el tiempo de pensar contra el número de transacciones que han utilizado ese tiempo. Esta gráfica se define en la cláusula 5.6.3 del TPC-C.
- Evolución del rendimiento a lo largo del periodo de test: representa el tiempo transcurrido contra el rendimiento obtenido por el sistema. Esta gráfica se define en la cláusula 5.6.4 del TPC-C.

2.2.2. Conjunto de Emuladores de Terminal Remoto (ETR)

Realizan las siguientes funciones:

- Emulación de usuario: generación de los datos de las transacciones según los perfiles de transacción definidos en la cláusula 2 del estándar TPC-C, selección de la transacción siguiente de acuerdo con las especificaciones de la cláusula 5.2.3 del estándar TPC-C y simulación de los tiempos de teclado y de pensar descritos en las cláusulas 5.2.5.3 y 5.2.5.4 del estándar TPC-C respectivamente.

- Emulación de terminal: muestra de los datos introducidos por el usuario emulado y los datos resultantes de la ejecución de la transacción.

- Envío de transacciones y recepción de resultados a través de los mecanismos de comunicación con el Monitor de Transacciones descritos en el apartado 2.2.6.

- Medida del tiempo de respuesta de transacción según se define en la cláusula 5.3.4 del estándar TPC-C.

- Registro en los ficheros de bitácora de los datos resultantes de la ejecución de la transacción necesarios para el posterior cálculo del rendimiento

El tiempo de respuesta de transacción que mide el ETR se define como el tiempo transcurrido desde que éste envía la transacción, hasta que recibe los resultados provenientes del Monitor de Transacciones. En el caso de la transacción Delivery este tiempo se corresponde con el tiempo que el ETR tarda en encolar la transacción.

Cada ETR está asociado a un almacén y un distrito de la base de datos.

El ETR recibe la señal de expiración del periodo de medida enviada por el Controlador del Benchmark para la finalización de su ejecución.

2.2.3. Monitor de Transacciones (MT)

El Monitor de Transacciones actúa como mediador entre el conjunto de terminales y el motor de la base de datos, encargándose de la recepción de las transacciones enviadas por los terminales y la ejecución de las mismas según su orden de llegada. El MT además, registra en las bitácoras correspondientes los datos resultantes de la ejecución aplazada de la transacción Delivery (según se especifica en la cláusula 2.7.2 del estándar TPC-C), así como los datos que permitirán calcular su tiempo de ejecución y los datos acerca de los posibles errores en la ejecución de las transacciones.

Tras recibir la señal de expiración del periodo de medida enviada por el controlador del benchmark el MT procede a su desactivación.

2.2.4. Controlador de Checkpoints

Es el encargado de la realización de checkpoints periódicos en la base de datos, según se describe en la cláusula 5.5.2.2 del estándar TPC-C, y del registro de los sellos de hora de comienzo y fin de los checkpoints. El primero de ellos se realiza inmediatamente después de su activación, que se produce al comienzo del intervalo de medida según se describe en el controlador del Benchmark.

2.2.5. Controlador de Limpiezas

En controlador de limpiezas se encarga de eliminar el efecto producido por la continua ejecución de operaciones en la base de datos. Por cada ejecución, postgreSQL mantiene información residual que relentiza dichas operaciones. Por ello será necesario efectuar limpiezas periódicas si no se quiere que caiga el rendimiento medido.

El Controlador de Limpiezas se encarga de realizar dichas limpiezas periódicamente sobre la base de datos, ejecutando el comando VACUUM [27] de postgreSQL.

El usuario puede elegir si desea realizar estas limpiezas periódicas. Si lo hace puede configurar el intervalo entre ellas así como su número máximo. La primera de las limpiezas se realizará una vez transcurrido el intervalo seleccionado tras al comienzo del test. El Controlador de Limpiezas no se activa durante el test si se decide no realizarlas.

2.2.6. Mecanismos de comunicación entre el conjunto de ETRs y el MT

La comunicación se establece utilizando los mecanismos de comunicación entre procesos IPC del UNIX System V [15]: semáforos, memoria compartida y colas de mensajes.

Semáforos. Un semáforo es un mecanismo que evita el conflicto entre dos o más procesos que solicitan simultáneamente un recurso compartido.

El semáforo únicamente indica la posibilidad de colisionar o no con otro proceso, pero no cierra el acceso al recurso compartido. Por lo tanto, si un proceso pretende acceder a dicho recurso compartido, atendiendo al semáforo puede evitar la colisión con otro proceso que intente acceder en el mismo instante.

Un semáforo es una estructura de tipo entero sobre la que se pueden realizar dos operaciones: una operación P que decrementa el valor del semáforo y una operación V que incrementa el valor del semáforo. Un semáforo no puede tomar un valor negativo, por lo que si una operación P provoca que el semáforo tenga un valor negativo, la operación se detiene hasta que el valor del semáforo sea lo suficientemente grande como para que el resultado sea positivo.

Las operaciones P y V deben ser atómicas, es decir, una operación P no puede ser interrumpida por otra operación P o V y viceversa. Esto garantiza que cuando varios procesos compitan con la adquisición de un semáforo, sólo uno de ellos consiga realizar la operación [16].

Memoria compartida. La memoria convencional a la que puede acceder un proceso es local a ese proceso y cualquier intento de direccionar esa memoria por otro proceso, provocará una violación de segmento. Algunos Sistemas Operativos como es el caso del UNIX System V brindan la posibilidad de crear zonas de memoria accesibles por varios procesos. Esta memoria va a ser virtual, por lo que sus direcciones físicas pueden variar con el tiempo. Esto no va a plantear ningún problema ya que los procesos generan direcciones virtuales y es el núcleo el encargado de transformarlas a direcciones físicas.

La memoria compartida proporciona un mecanismo rápido de comunicación entre procesos. Si un proceso desea enviar cierta información a otro proceso, tan sólo debe escribir los datos en la memoria compartida e inmediatamente el proceso destinatario puede acceder a dichos datos [16].

Colas de mensajes. Una cola es una estructura gestionada por el kernel donde van a poder escribir varios procesos. Los mecanismos de sincronismo para que no se produzcan colisiones son responsabilidad del núcleo. Los datos que se escriben en la cola deben tener un formato de mensaje definido y son tratados como un todo indivisible.

Puede haber varios procesos introduciendo datos en la cola, al igual que puede haber otros tantos que extraigan esos datos. La operación de escritura va a introducir un mensaje y la de lectura va a extraer un mensaje de la cola. La cola se gestiona como un mecanismo FIFO donde el primer mensaje que entra es el primer mensaje que sale, aunque para dotarla de más flexibilidad se pueden hacer peticiones de lectura para extraer un mensaje de un tipo determinado, con lo que se rompe la gestión de tipo fifo aunque se sigue manteniendo para los mensajes del mismo tipo. Si no se especifica un tipo, se recoge el primer mensaje de la cola [16].

Etapas de la comunicación

Como se puede apreciar en la figura 2.3, los ETR se comunican con el MT a través de dos canales unidireccionales: la memoria compartida y la cola de

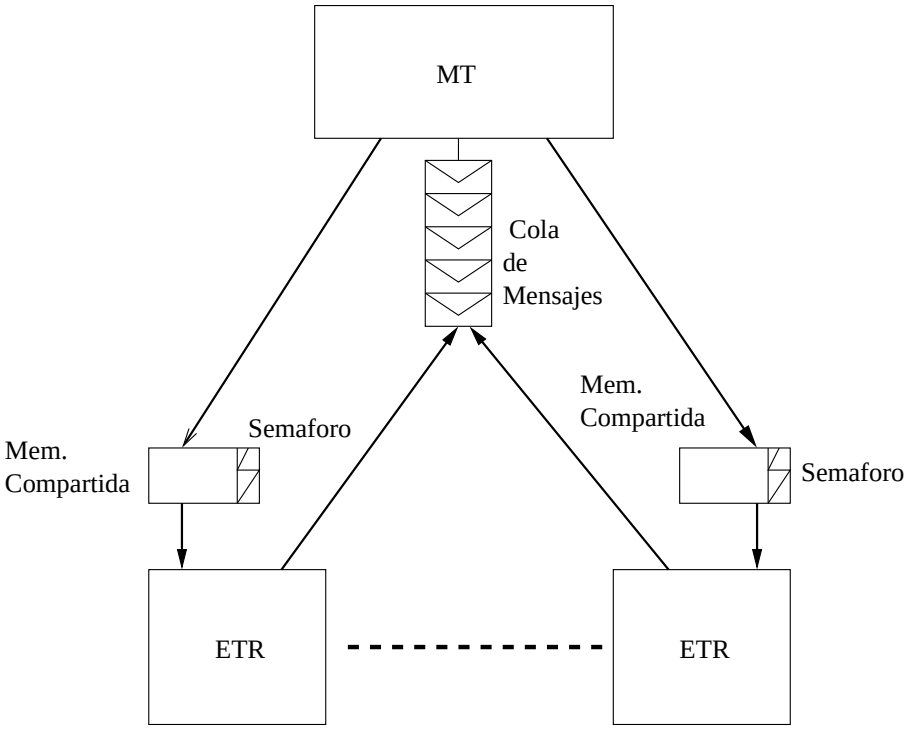


Figura 2.3: Comunicación ETR - MT

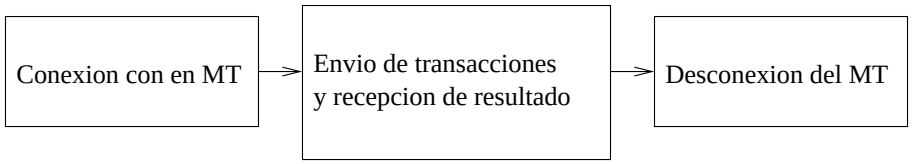


Figura 2.4: Etapas de la comunicación ETR - MT

mensajes, que están sincronizados por medio de un semáforo. Los mensajes de información que tienen como destino el MT se envían a través de la cola de mensajes creada por el MT y por el contrario los mensajes que tienen como destino el ETR se envían a través de la memoria compartida que cada ETR ha creado. El ETR envía un nuevo mensaje a través de la cola de mensajes, y cierra el semáforo de sincronismo. Cuando el MT recibe el mensaje y termina de procesarlo, abre el semáforo permitiendo al ETR el acceso a los resultados.

La figura 2.4 ilustra el ciclo de comunicaciones ETR - MT.

Etapa de Conexión

Inicialmente los canales de comunicación no existen, por lo que es necesario que el ETR informe al MT que quiere comunicarse con el, a ese proceso se le denomina

conexión. En el proceso de conexión el MT enlaza con la memoria compartida y con el semáforo de sincronismo que el ETR ha creado. La solicitud de conexión se realiza enviando un mensaje de conexión a través de la cola de mensajes que el MT crea al activarse.

Envío de Transacciones y Recepción de Resultados

Las transacciones se envían utilizando los canales que se han creado tras la conexión. El envío de transacciones sigue el esquema del gráfico que ilustra el ciclo de comunicaciones ETR-MT. Para recibir los resultados el ETR espera a que se abra el semáforo de sincronismo, para leer de la memoria compartida.

El ETR permanece en esta etapa mientras el Controlador del Benchmark no le indique que el test ha finalizado.

Etapa de Desconexión

Cuando los ETRs reciben la señal de expiración del periodo de test deben eliminar la memoria compartida y los semáforos de sincronismo asociados. Para poder eliminar la memoria compartida y los semáforos de forma eficiente, es necesario que el MT se desenganche de la memoria compartida y posteriormente el ETR indique al sistema operativo que la libere. A este proceso se le denomina desconexión. La solicitud de desconexión se realiza enviando un mensaje de desconexión a través de la cola de mensajes.

2.3. Implementación del software

En este apartado se hará una descripción detallada del funcionamiento de cada uno de los módulos que implementan este benchmark, así como de los mecanismos que se han utilizado para la realización de las funciones específicas de cada uno de ellos.

2.3.1. Consideraciones generales de la implementación

2.3.1.1 Gestor de dase de datos

El gestor de base de datos utilizado para implementar el benchmark es postgreSQL 7.1.3 [24]. El mecanismo utilizado para la comunicación con la base de datos consiste en incrustar sentencias en lenguaje SQL dentro del código C del

benchmark, y preprocesarlas con el programa *ecpg* [26, 27] que proporciona postgreSQL y que transforma las sentencias en funciones de C que interactúan directamente con el gestor de base de datos. En este apartado se describe cómo se ha utilizado este gestor para implementar los requisitos del estándar TPC-C.

Tipos de datos. A continuación se describen los tipos de datos definidos por postgreSQL que se han utilizado para implementar la base de datos [28].

1. `int2` representa un entero con signo de 2 bytes.
2. `int4` representa un entero con signo de 4 bytes.
3. `float4` representa un real de simple precisión de 4 bytes.
4. `float8` representa un real de doble precisión de 4 bytes.
5. `numeric(n,m)` representa un real con `n` cifras enteras y `m` decimales.
6. `char(n)` representa una cadena de n caracteres con longitud fija.
7. `varchar(n)` representa una cadena de caracteres de longitud variable con tamaño máximo n.
8. `timestamp` representa una cadena capaz de almacenar fechas y horas.

Conexión con la base de datos. Los programas que interactúan con la base de datos deben conectar con postgreSQL antes de realizar cualquier consulta [27]. El comando SQL definido por postgreSQL para la conexión es:

```
EXEC SQL CONNECT TO 'base' USER 'usuario';
```

donde:

- `'base'` es el nombre de la base de datos a la que se conecta.
- `'usuario'` es el nombre de usuario con permisos para acceder a la base de datos a la que se pretende conectar.

En adelante, la expresión *conexión con la base de datos <nombre>* significa que se ejecuta la sentencia anterior para la base de datos *<nombre>* y nombre de usuario *postgres* que es el usuario que se crea al instalar postgreSQL.

Tras todas las conexiones se activa el modo de ejecución *autocommit* [27]. En este modo de ejecución todas las operaciones simples se confirmarán automáticamente. Un grupo de operaciones se dice que está *confirmado* (*commited* cuando

sus resultados se ven reflejados en la base de datos. Hasta que una operación no se confirma, los resultados no se vuelcan en la base de datos. Para que un grupo de operaciones se considere dentro de una transacción la primera operación del grupo debe estar precedida por la sentencia de SQL `EXEC SQL BEGIN`.

El hecho de que únicamente se considere que ha comenzado una transacción mediante la sentencia anterior, permite realizar operaciones que requieren que su ejecución esté fuera de un bloque de transacción. Tales operaciones son, por ejemplo, crear o eliminar una base de datos.

El modo de ejecución *autocommit* se activa por medio de la sentencia:

```
EXEC SQL SET autocommit = ON;
```

Implementación de transacciones. Para implementar las transacciones que especifica el TPC-C todas las operaciones recogidas en sus perfiles de transacción (cláusula 2), se enmarcan dentro de las sentencias `EXEC SQL BEGIN` y `EXEC SQL COMMIT` si la transacción se confirma o `EXEC SQL ROLLBACK` si la transacción se cancela.

Al enmarcar un conjunto de operaciones dentro de una transacción, se consigue que la ejecución del conjunto sea atómica, es decir que todo el conjunto se considere como una única operación, evitando por tanto, ejecuciones parciales. Las operaciones dentro de una transacción no se reflejan en la base de datos hasta que no se confirme. Si se cancela la transacción las operaciones previas no producen efecto en la base de datos [18].

Checkpoints. PosgreSQL, para aumentar la velocidad de las consultas, mantiene unos ficheros secuenciales que contienen las modificaciones que se han realizado sobre la base de datos. De esta forma, cada vez que se produce una modificación, se escribe en los ficheros secuenciales en vez de realizarla directamente en la base de datos. El acceso a un fichero secuencial es más rápido que la modificación de la estructura de datos que mantiene las tablas.

PostgreSQL periódicamente libera la información de los ficheros secuenciales trasladando las modificaciones a la base de datos. A esta operación se la conoce como *checkpoint* [25].

PostgreSQL realiza un checkpoint automático si los ficheros secuenciales se han llenado o si el periodo de tiempo máximo entre dos checkpoints automáticos ha expirado.

Para cumplir las especificaciones del TPC-C en cuanto a la realización de checkpoints (cláusula 5.5.2.2), el módulo Controlador de Checkpoints los realiza periódicamente sobre la base de datos, cada 30 minutos. La sentencia SQL que permite

realizar un checkpoint es la siguiente:

```
EXEC SQL CHECKPOINT;
```

Es necesario evitar los checkpoints que postgreSQL realiza automáticamente, ya que pueden suceder entre dos checkpoints forzados por el Controlador. Para ello se han modificado los siguientes parámetros de la configuración de postgreSQL:

`wal_files` indica el numero de ficheros secuenciales que se crean al arrancar postgreSQL. Se ha fijado este parámetro al valor 10.

`checkpoint_segments` indica el numero de ficheros secuenciales que deben llenarse antes de hacer un checkpoint. Se ha fijado al valor 10.

`checkpoint_timeout` indica el tiempo máximo, expresado en segundos entre dos checkpoints automaticos. Se ha fijado al valor 3600.

Estos parámetros se encuentran en el fichero de configuración de postgreSQl `postgresql.conf` [25] que se encuentra en el directorio `/usr/local/pgsql/data`.

Con la configuración anterior, se reduce en gran medida la posibilidad de que postgreSQL realice un checkpoint antes que el Controlador.

Limpiezas de base de datos. Para eliminar los datos residuales resultantes de la ejecución intensiva de operaciones sobre la base de datos, se utiliza el comando SQL de postgreSQL:

```
EXEC SQL VACUUM ANALYZE
```

Este comando además actualiza las tablas de estadísticas utilizadas por el optimizador de consultas de postgreSQL para encontrar el modo más eficiente para ejecutar una consulta.

Volcado en disco. Por defecto, postgreSQL vuelca los datos inmediatamente al disco ya sea en la base de datos o en los ficheros secuenciales, cada vez que realiza una transacción. El rendimiento mejora si se retarda esa escritura, sin embargo, la consistencia de la base de datos puede verse afectada en caso de que el sistema falle. Para que posgreSQL retarde la escritura de los datos de las transacciones es necesario desactivar el parámetro `fsync` [25].

Tratamiento de errores en SQL. Tras las operaciones que se realizan contra la base de datos se comprueba si ha habido algún error mediante la estructura

`sqlca` [27]. Para poder utilizar esta estructura se incluye en las secciones de ficheros de cabecera de los módulos la siguiente línea:

```
exec sql include sqlca;
```

Si después de ejecutar una sentencia SQL el campo `sqlca.sqlcode` contiene un valor distinto de cero, indica que se ha producido un error. Es ese caso el campo `sqlca.sqlerrm.sqlerrmc` contendrá una descripción del error. La formato de la estructura `sqlca` se explica en el apéndice C.

Algunos valores del campo `sqlca.sqlcode` se utilizan para determinar circunstancias y tomar decisiones en la ejecución de las transacciones. Este es el caso, por ejemplo, del código 100 que indica que en la anterior selección de fila no se ha encontrado ninguna fila coincidente.

2.3.1.2 Implementación de la base de datos

En este apartado se explicará cómo se ha implementado la base de datos, para cumplir con los requisitos de las especificaciones del estándar TPC-C.

Se explicará cómo se ha creado el esquema de cada una de las nueve tablas, y qué tipo de datos se ha asignado a cada uno de sus atributos para cumplir con la cláusula 1 del TPC-C.

También se explicara cómo se generan cada uno de los valores que asignan a cada uno de los atributos de las nueve tablas en la población inicial (población previa al test de rendimiento) para cumplir con los requisitos de la cláusula 4 del TPC-C.

Además se detallarán cada una de las llaves primarias y llaves foráneas que se han declarado en cada una de las tablas, según se especifica en la cláusula 1 del TPC-C, y que aseguran el mantenimiento de la integridad referencial.

La creación y la población de las nueve tablas corre a cargo del Controlador del Benchmark. El tamaño de la base de datos a crear lo determina el usuario, y viene determinado por el número de almacenes (*warehouse*). El número de máximo de almacenes con el que puede trabajar el TPCC-UVA se define como constante en el fichero de cabecera `tpcc.h` y está fijado a 100.

La base de datos que se utiliza en el benchmark se llama `tpcc`. Para la creación de las nueve tablas de la base de datos, se utiliza la función `creartablas()`. En la explicación de cada una de las tablas de detallarán las funciones que se han utilizado para su población.

Se han respetado los nombres originales tanto de las tablas como de sus atributos para facilitar la comprensión a la hora de consultar el estándar TPC-C.

El estándar TPC-C especifica que los valores de algunos de los campos de las tablas han de generarse aleatoriamente, siguiendo una distribución uniforme. Para cumplir con este requisito, a cada variable aleatoria con distribución uniforme, se le asigna un vector de estado. De los mecanismos para la generación de números aleatorios se hablará en la sección 2.3.1.3.

Para la generación de los valores de algunos de los atributos de las nueve tablas se utilizan las siguientes funciones definidas en el fichero de cabecera `tpcc.h`:

- `aleat_int()`: genera un número entero aleatorio comprendido comprendido entre [x, y] con una media de (x+y)/2.
- `cad_alfa_num()`: genera una cadena de caracteres alfanuméricos aleatorios de tamaño, también aleatorio, comprendido entre [x,y] con una media de (x+y)/2.
- `aleat_dbl()`: genera un número aleatorio en coma flotante comprendido entre [x,y] con una media de (x+y)/2.
- `cad_num()`: genera una cadena de caracteres numéricos aleatorios de tamaño N.
- `crea_clast()`: genera el campo `C_LAST` a partir de un número que se le pasa como parámetro según se especifica en la cláusula 4.3.2.3.
- `getfechahora()`: genera una cadena de caracteres que contiene la fecha y la hora actual del sistema.
- `aleat_vect()`: genera N números aleatorios y les ordena en un vector por orden ascendente.
- `permutacion_int()`: genera una permutación en un vector de tamaño N, de los números naturales entre [1,N].
- `nurand()`: genera un número aleatorio entre [x,y] con distribución no uniforme, según se especifica en la cláusula 2.1.6 del estándar TPC-C.

El tamaño de cada una de las nueve tablas que componen la base de datos, está definido como constante en el fichero de cabecera `tpcc.h`. Los valores de estas constantes se corresponden con los tamaños de las tablas que se especifican en la cláusula 4 del TPC-C.

A continuación se describe el esquema de las nueve tablas, mediante la sentencia SQL que se ha utilizado para su creación, así cómo los métodos utilizados para la generación de los valores de cada atributo en la población inicial.

Tabla Warehouse

El esquema de la tabla `warehouse` (o tabla de Almacén) es el siguiente.

```
EXEC SQL CREATE TABLE warehouse (
     w_id int4,
     w_name varchar(10),
     w_street_1 varchar(20),
     w_street_2 varchar(20),
     w_city varchar(20),
     w_state char(2),
     w_zip char(9),
     w_tax float4,
     w_ytd float8,
CONSTRAINT wareh1 PRIMARY KEY (w_id)
 );
```

En esta tabla se define como llave primaria el campo `w_id` (identificador de almacén). Esto restringe la posibilidad de insertar una fila con un identificador de almacén ya existente en la base de datos.

Con el atributo declarado como llave primaria postgreSQL crea automáticamente un índice para acelerar las búsquedas en la tabla `warehouse`.

Para la población inicial de la tabla `warehouse` se utiliza la función `poblar_warehouse()`, y se inserta una fila por cada uno de los almacenes seleccionados por el usuario.

La cardinalidad de esta tabla no variará durante el test.

Los valores asignados a cada uno de los atributos de esta tabla en la población inicial, para cumplir con los requisitos de la cláusula 4.3 del TPC-C son:

- `w_id`: se le asigna un número consecutivo a medida que se insertan filas en la tabla.
- `w_name`: se le asigna una cadena alfanumérica de caracteres aleatorios de tamaño comprendido entre 6 y 10, mediante la función `cad_alfa_num()`.
- `w_street_1, w_street_2, w_city`: se les asigna una cadena alfanumérica de caracteres aleatorios de tamaño comprendido entre 10 y 20, mediante la función `cad_alfa_num()`.
- `w_state`: se le asigna una cadena alfanumérica de dos caracteres aleatorios mediante la función `cad_alfa_num()`.
- `w_zip`: se le asigna la cadena formada por la concatenación de las cadenas '11111' y la cadena aleatoria de cuatro caracteres numéricos generada mediante la función `cad_num()`.

- `w_tax`: se le asigna un número real aleatorio comprendido entre 0 y 0,2 mediante la función `aleat_dbl()`.

- `w_ytd`: se le asigna el valor 300.000.

Tabla Item

El esquema de la tabla `item` (o tabla de artículos) es el siguiente.

```
EXEC SQL CREATE TABLE item (
     i_id int4,
     i_im_id int4,
     i_name varchar(24),
     i_price float8,
     i_data varchar(50),
CONSTRAINT item1 PRIMARY KEY (i_id)
)
```

En esta tabla se declara como llave primaria al campo `i_id` (identificador de artículo). Esto impide la posibilidad de que se inserten dos filas con el mismo identificador de artículo.

Con el atributo declarado como llave primaria postgreSQL crea automáticamente un índice para acelerar las búsquedas en la tabla `item`.

Para la población de esta tabla se utiliza la función `poblar_item()`, y se insertan 100.000 filas independientemente del número de almacenes seleccionados.

La cardinalidad de la tabla `item` no variará durante el test.

Los métodos para la población inicial de esta tabla, usados para cumplir los requisitos de la cláusula 4.3 del TPC-C, son:

- `i_id`: se le asigna un número consecutivo a medida que se insertan las filas.

- `i_im_id`: se le asigna un número entero aleatorio entre 1 y 10.000 mediante la función `aleat_int()`.

- `i_name`: se le asigna una cadena alfanumérica de tamaño comprendido entre 14 y 24 mediante la función `cad_alfa_num()`.

- `i_price`: se le asigna un número real aleatorio comprendido entre 1,00 y 100,00 mediante la función `aleat_dbl()`.

- `i_data`: el 10 % de los artículos han de contener la cadena 'ORIGINAL' en una posición aleatoria. Para ello primero se le asigna a este atributo una cadena alfanumérica aleatoria de tamaño comprendido entre 26 y 50 mediante la función `cad_alfa_num()`. Después se determina si la cadena debe contener la palabra 'ORIGINAL'. Para ello se comprueba si el identificador del artículo (`i_id`) está contenido en un vector de número aleatorios únicos con un tamaño 10 veces menor que el número total de artículos, generado mediante la función `aleat_vect()`.

Tabla stock

El esquema de la tabla `stock` es el siguiente:

```
EXEC SQL CREATE TABLE stock (
     s_i_id int4,
     s_w_id int4,
     s_quantity int2,
     s_dist_01 char(24),
     s_dist_02 char(24),
     s_dist_03 char(24),
     s_dist_04 char(24),
     s_dist_05 char(24),
     s_dist_06 char(24),
     s_dist_07 char(24),
     s_dist_08 char(24),
     s_dist_09 char(24),
     s_dist_10 char(24),
     s_ytd numeric(8,2),
     s_order_cnt int2,
     s_remote_cnt int2,
     s_data varchar(50),
CONSTRAINT stock1 PRIMARY KEY (s_w_id, s_i_id),
CONSTRAINT stock2 FOREIGN KEY (s_w_id)
        REFERENCES warehouse (w_id) DEFERRABLE,
CONSTRAINT stock3 FOREIGN KEY (s_i_id)
        REFERENCES item (i_id) DEFERRABLE
)
```

En esta tabla se definen como llave primaria a los campos `s_w_id` y `s_i_id` lo que impedirá insertar dos filas con esos dos campos iguales.

Con los atributos declarados como llave primaria, postgreSQL crea automáticamente un índice para acelerar las búsquedas en esta tabla.

También se definen dos llaves foráneas. La primera corresponde al campo `s_w_id`, que referencia al campo `w_id`. Esto impedirá insertar cualquier fila en la tabla `stock` cuyo valor de `s_w_id` no sea ninguno de los valores de `w_id` de las filas de la tabla `warehouse`, es decir, todas las filas de la tabla stock han de pertenecer a almacenes que aparezcan reflejados en la base de datos. La otra llave foránea la constituye el campo `s_i_id` que referencia al campo `i_id` de la tabla `item`. Esto impedirá que se inserten filas en la tabla `stock` cuyo valor de la llave foránea no aparezca en el campo `i_id`, es decir, no se podrán insertar filas en la tabla `stock` de artículos que no aparezcan en la tabla `item`

Para la población inicial de esta tabla se utiliza la función `poblar_stock` y se insertan 100.000 por cada uno de los almacenes seleccionados.

La cardinalidad de esta tabla no variará durante el test.

Los métodos para la población inicial de esta tabla, usados para cumplir los requisitos de la cláusula 4.3 del TPC-C, son:

- `s_i_id`: se le asigna un entero consecutivo a medida que se insertan las filas. para cada uno de los almacenes.
- `s_w_id`: se le asigna un entero consecutivo por cada uno de los almacenes.
- `s_quantity`: se le asigna un número entero aleatorio entre 10 y 100 mediante la función `aleat_int()`.
- `s_dist_01 ... s_dist_10`: se les asigna una cadena numérica aleatoria de 24 caracteres, mediante la función `cad_alfa_num()`.
- `s_ytd`: se le asigna el valor 0.
- `s_order_cnt`: se le asigna el valor 0.
- `s_data`: se utiliza el mismo método que para el campo `i_data`.

Tabla District

El diseño de la tabla `district` (o tabla de distrito) es el siguiente:

```
EXEC SQL CREATE TABLE district (
     d_id int4,
     d_w_id int4,
     d_name varchar(10),
     d_street_1 varchar(20),
     d_street_2 varchar(20),
     d_city varchar(20),
```

```
    d_state char(2),
    d_zip char(9),
    d_tax float4,
    d_ytd float8,
    d_next_o_id int4,
CONSTRAINT dist1 PRIMARY KEY (d_w_id,d_id),
CONSTRAINT dist2 FOREIGN KEY (d_w_id)
        REFERENCES warehouse (w_id) DEFERRABLE
)
```

En esta tabla se definen como llave primaria los atributos `d_w_id` y `d_id` lo que impedirá que se inserten dos filas con estos dos campos iguales.

Con los atributos declarados como llave primaria postgreSQL crea automáticamente un índice para acelerar las búsquedas en la tabla `district`.

Además se define como llave foránea al atributo `d_w_id` referenciando al campo `w_id`. Esto impedirá que se inserten filas en la tabla `district` cuyo valor de llave foránea, no aparezca en el campo `w_id`, es decir, no se podrán insertar filas de distritos que pertenezcan a almacenes que no aparezcan en la tabla `warehouse`.

En la tabla `district` se insertan 10 filas por cada almacén seleccionado por el usuario. Para su población se utiliza la función `poblar_district()`.

Los métodos para la población inicial de esta tabla, usados para cumplir los requisitos de la cláusula 4.3, son:

- `d_id`: se le asigna un entero consecutivo a medida que se insertan las filas para cada uno de los almacenes.
- `d_w_id`: se le asigna el identificador del almacén al que pertenece.
- `d_name`: se le asigna una cadena alfanumérica de tamaño comprendido entre 6 y 10 mediante la función `cad_alfa_num()`.
- `d_street_1, d_street_2`: se le asigna una cadena alfanumérica de tamaño comprendido entre 10 y 20 mediante la función `cad_alfa_num()`.
- `d_city`: se le asigna una cadena alfanumérica de tamaño comprendido entre 10 y 20 mediante la función `cad_alfa_num()`.
- `d_state`: se le asigna una cadena alfanumérica de dos caracteres mediante la función `cad_alfa_num()`.
- `d_zip`: se le asigna la cadena formada por la concatenación de las cadenas '11111' y la cadena aleatoria de cuatro caracteres numéricos generada mediante la función `cad_num()`.

- `d_tax`: se le asigna un número real aleatorio comprendido entre 0 y 0,2 mediante la función `aleat_dbl()`
- `d_ytd`: se le asigna el valor 30.000.
- `d_next_o_id`: se le asigna el valor 3001.

Tabla Customer

El esquema de la tabla `customer` (o tabla de cliente) es el siguiente:

```
EXEC SQL CREATE TABLE customer (
     c_id int4,
     c_d_id int4,
     c_w_id int4,
     c_fi rst varchar(16),
     c_middle char(2),
     c_last varchar(16),
     c_street_1 varchar(20),
     c_street_2 varchar(20),
     c_city varchar(20),
     c_state char(2),
     c_zip char(9),
     c_phone char(16),
     c_since timestamp DEFAULT '1-1-1970',
     c_credit char(2),
     c_cre dit_lim float8,
     c_discount float4,
     c_balance float8,
     c_ytd_payment float8,
     c_payment_cnt int2,
     c_delivery_cnt int2,
     c_data varchar(500),
CONSTRAINT custom1 PRIMARY KEY (c_w_id, c_d_id, c_id),
CONSTRAINT custom2 FOREIGN KEY (c_w_id, c_d_id)
         REFERENCES district (d_w_id, d_id) DEFERRABLE
);
```

En esta tabla se definen los atributos `c_w_id, c_d_id` y `c_id` como llave primaria lo que impedirá que se inserten dos filas con esos dos campos iguales.

Con los atributos declarados como llave primaria postgreSQL crea automáticamente un índice para acelerar las búsquedas en esta tabla.

Además se definen como llave foránea los atributos `c_w_id, c_d_id` referenciando a `d_w_id, d_id`. Esto impedirá que se inserten filas cuyos valores de la

llave foránea no aparezcan en los campos `d_w_id` y `d_id`, es decir, no se podrán insertar filas en la tabla customer que pertenezcan a distritos y a almacenes que no aparezcan en las tablas `district` y `warehouse` respectivamente.

Para la población inicial de la tabla `customer` se utiliza la función `poblar_customer()`, insertándose 3000 filas por cada distrito, por lo tanto 30.000 por cada almacén seleccionado por el usuario.

La cardinalidad de esta tabla no variará durante el test.

Los métodos para la población inicial de esta tabla, usados para cumplir los requisitos de la cláusula 4.3, son:

- `c_id` se le asigna un entero consecutivo a medida que se van insertando filas, para cada almacén y para cada distrito.
- `c_d_id` se le asigna el valor del distrito al que pertenece el cliente
- `c_w_id` se le asigna el valor del almacén al que pertenece el cliente.
- `c_first` se le asigna una cadena alfanumérica aleatoria de tamaño comprendido entre [8, 10] mediante la función `cad_alfa_num()`.
- `c_middle` se le asigna la cadena 'OE'.
- `c_last` para cada distrito, este campo se genera por medio de la función `crea_clast()`. En las 1000 primeras filas se utiliza consecutivamente un número de 0 a 999. En las 2000 filas restantes se utiliza un número aleatorio generado mediante la función `nurand()` a partir de un número aleatorio constante (C_LOAD) durante toda la carga, como se define en la cláusula 2.1.6 del TPC-C.
- `c_street_1, c_street_2, c_city` se les asigna una cadena alfanumérica aleatoria de tamaño comprendido entre [10, 20] mediante la función `cad_alfa_num()`.
- `c_state` se le asigna una cadena alfanumérica aleatoria de dos caracteres.
- `c_zip` se le asigna la cadena formada por la concatenación de las cadenas '11111' y la cadena aleatoria de cuatro caracteres numéricos generada mediante la función `cad_num()`.
- `c_phone` se le asigna una cadena aleatoria de 16 dígitos generada mediante la función `cad_num()`
- `c_since` se le asigna la fecha del sistema obtenida mediante la función `getfechahora()`.

- `c_credit` para el 10 % de las filas en cada distrito se introduce el valor `BC`, para el 90 % restante se introduce `GC`. El 10 % de las filas se discrimina mediante un vector aleatorio de tamaño 10 veces menor que el número de clientes del distrito. Si el campo `c_id` del cliente coincide con uno de los valores contenidos en el vector, se inserta el valor `BC`.
- `c_credit_lim` se fija a 50.000.
- `c_discount` se genera un número real aleatorio entre 0 y 0,5 mediante la función `aleat_dbl()`.
- `c_balance` se fija a -10.
- `c_ytd_payment` se fija a 10.
- `c_payment_cnt` se fija a 1.
- `c_delivery_cnt` se fija a 0.
- `c_data varchar` se genera una cadena aleatoria de tamaño entre 300 y 500 mediante la función `cad_alfa_num()`.

Tabla History

El esquema de la tabla `history` (o tabla de histórico) es el siguiente:

```
EXEC SQL CREATE TABLE history (
     h_c_id int4,
     h_c_d_id int4,
     h_c_w_id int4,
     h_d_id int4,
     h_w_id int4,
     h_date timestamp4 DEFAULT '1970-01-01',
     h_amount float4,
     h_data varchar(24),
CONSTRAINT hist1 FOREIGN KEY (h_c_w_id, h_c_d_id, h_c_id)
         REFERENCES customer (c_w_id, c_d_id, c_id) DEFERRABLE,
CONSTRAINT hist2 FOREIGN KEY (h_w_id, h_d_id)
         REFERENCES district (d_w_id, d_id) DEFERRABLE
);
```

En esta tabla se definen dos llaves foráneas. La primera está compuesta por los atributos `h_c_w_id, h_c_d_id, h_c_id` referenciando a `c_w_id, c_d_id, c_id`. Esto impedirá que se inserten filas en la tabla `history` cuyos campos de llave foránea no aparezcan en los campos referenciados. La segunda está compuesta por los atributos `h_w_id, h_d_id` referenciando a los campos `d_w_id, d_id`, que

también impedirá que se inserten filas con valores de llave foránea distintos a los referenciados.

En la tabla `history` se insertan 3000 filas por cada distrito, por lo tanto 30.000 por almacén seleccionado por el usuario. Para su población se utiliza la función `poblar_history()`.

La cardinalidad de esta tabla variará durante el test ya que se inserta una nueva fila en la tabla por cada transacción Payment ejecutada.

Los métodos para la población inicial de esta tabla, usados para cumplir los requisitos de la cláusula cuatro, son:

- `h_c_id` se le asigna un entero consecutivo a medida que se van insertando filas, para cada almacén y para cada distrito.
- `h_c_d_id` y `h_d_id` se les asigna el valor del destrito al que pertenece el cliente.
- `h_c_w_id` y `h_w_id` se les asigna el valor del almacén al que pertenece el cliente.
- `h_date` se introduce la fecha del sistema obtenida mediante la función `getfechahora()`.
- `h_amount` se le asigna el valor 10.
- `h_data` se le asigna una cadena alfanumérica aleatoria de tamaño comprendido entre 12 y 24 mediante la función `cad_alfa_num()`.

Tabla Orderr.

El esquema de la tabla `orderr` (o tabla de órdenes) es el siguiente:

```
EXEC SQL CREATE TABLE orderr (
     o_id int4,
     o_w_id int4,
     o_d_id int4,
     o_c_id int4,
     o_entry_d timestamp DEFAULT '1970-01-01',
     o_carrier_id int2 DEFAULT 0,
     o_ol_cnt int2,
     o_all_local int2,
CONSTRAINT orderr1 PRIMARY KEY (o_w_id, o_d_id, o_id),
CONSTRAINT orderr2 FOREIGN KEY (o_w_id, o_d_id, o_c_id)
         REFERENCES customer (c_w_id, c_d_id, c_id) DEFERRABLE
);
```

NOTA: se ha alterado el nombre de esta tabla con respecto al nombre propuesto en el estándar TPC-C para evitar conflictos con el comando de postgreSQL `ORDER`.

En esta tabla se definen los atributos `o_w_id, o_d_id` y `o_id` como llave primaria lo que impedirá que se inserten dos filas con esos tres capos iguales.

Con los atributos declarados como llave primaria postgreSQL crea automáticamente un índice para acelerar las búsquedas en esta tabla.

Se define, además, una llave foránea compuesta por los atributos `o_w_id, o_d_id, o_c_id` referenciando a `c_w_id, c_d_id, c_id`. Esto impedirá que se inserten filas en la tabla `orderr` cuyos campos de llave foránea no aparezcan en los campos referenciados.

En la tabla `orderr` se insertan 3000 filas por cada distrito, por lo tanto 30.000 por almacén seleccionado por el usuario. Para su población se utiliza la función `poblar_orderr()`.

Durante el test, la cardinalidad de esta tabla no es fija ya que se insertará una nueva fila por cada transacción New-Order ejecutada.

Los métodos para la población inicial de esta tabla, usados para cumplir los requisitos de la cláusula 4.3 del TPC-C son:

- `o_id` se le asigna un entero consecutivo a medida que se van insertando filas, para cada almacén y para cada distrito.
- `o_w_id` se le asigna el valor del almacén al que pertenece la orden.
- `o_c_id` se les asigna el valor del cliente al que pertenece la orden.
- `o_d_id` se les asigna el valor del distrito al que pertenece la orden.
- `o_entry_d` se les asigna la fecha y la hora del sistema mediante la función `getfechahora()`.
- `o_carrier_id` en las 2100 primeras líneas de cada distrito, se les asigna un número aleatorio entre 1 y 10 mediante la función `aleat_int()`. A las 900 filas restantes se les asigna el valor 0.
- `o_ol_cnt` se le asigna un número aleatorio entre 5 y 10 mediante `aleat_int()`. Esta cifra determinará el número de filas de la tabla `order_line` por cada fila de la tabla `orderr`.
- `o_all_local` se le asigna el valor 1.

Tabla Order-Line

El esquema de la tabla `order_line` (o tabla de Líneas de Orden) es el siguiente:

```
EXEC SQL CREATE TABLE order_line (
     ol_o_id int4,
     ol_d_id int4,
     ol_w_id int4,
     ol_number int2,
     ol_i_id int4,
     ol_supply_w_id int4,
     ol_delivery_d timestamp DEFAULT '1970-01-01',
     ol_quantity int2,
     ol_amount numeric(6,2),
     ol_dist_info char(24),
CONSTRAINT ol1 PRIMARY KEY (ol_w_id, ol_d_id, ol_o_id, ol_number),
CONSTRAINT ol2 FOREIGN KEY (ol_w_id, ol_d_id, ol_o_id)
         REFERENCES orderr (o_w_id, o_d_id, o_id) DEFERRABLE,
CONSTRAINT ol3 FOREIGN KEY (ol_supply_w_id, ol_i_id)
         REFERENCES stock (s_w_id, s_i_id) DEFERRABLE
);
```

En esta tabla se definen los atributos `ol_w_id, ol_d_id, ol_o_id` y `ol_number` como llave primaria lo que impedirá que se inserten dos filas con esos cuatro campos iguales.

Con los atributos declarados como llave primaria postgreSQL crea automáticamente un índice para acelerar las búsquedas en la tabla `order_line`.

Se definen, además, dos llaves foráneas. La primera está compuesta por los atributos `ol_w_id, ol_d_id, ol_o_id` referenciando a `o_w_id, o_d_id, o_id`. Esto impedirá que se inserten filas en la tabla `orderr` cuyos campos de llave foránea no aparezcan en los campos referenciados. La segunda está compuesta por los atributos `ol_supply_w_id` y `ol_i_id`, que referencian a los campos `s_w_id` y `s_i_id`.

En la tabla `order_line` se insertan por cada una de las filas de la tabla `orderr` el número de filas indicado por el campo `o_ol_cnt` de la fila insertada en la tabla `orderr`. Para su población se también se utiliza la función `poblar_orderr()`.

Durante el test, la cardinalidad de esta tabla no es fija ya que por cada transacción New-Order ejecutada se insertan un número de líneas igual al campo `o_ol_cnt` de la fila insertada en la tabla `orderr`.

Los métodos para la población inicial de esta tabla, usados para cumplir los requisitos de la cláusula 4.3, son:

- `ol_o_id` se le asigna el número de orden al que pertenece la línea de orden
- `ol_d_id` se le asigna el valor del distrito al que pertenece la línea de orden.

- `ol_w_id` se les asigna el valor del almacén al que pertenece la línea orden.
- `ol_number` se le asigna un número consecutivo a medida que se insertan las líneas de artículo pertenecientes a la misma orden.
- `ol_i_id` se le asigna un entero aleatorio comprendido entre 1 y 100.000 mediante la función `aleat_int`.
- `ol_supply_w_id` se le asigna el valor del almacén al que pertenece la línea de orden.
- `ol_delivery_d`, en las líneas de orden pertenecientes a las 2100 primeras órdenes se iguala al atributo `o_entry_d`, el resto se le asigna la fecha por defecto.
- `ol_quantity` se le asigna el valor 5.
- `ol_amount` en las líneas de orden pertenecientes a las 2100 primeras órdenes se asigna el valor 0. En las restantes se le asigna un valor entre 0,01 y 9999,99 mediante la función `aleat_dbl()`
- `ol_dist_info` se le asigna un cadena alfanumérica de 24 caracteres mediante la función `cad_alfa_num()`.

Tabla New-Order.

El esquema de la tabla `new_order` (tabla de transacciones New-Order sin repartir) es el siguiente:

```
EXEC SQL CREATE TABLE new_order (
     no_o_id int4,
     no_d_id int4,
     no_w_id int4,
CONSTRAINT no1 PRIMARY KEY (no_w_id, no_d_id, no_o_id),
CONSTRAINT no2 FOREIGN KEY (no_w_id, no_d_id, no_o_id)
         REFERENCES orderr (o_w_id, o_d_id, o_id) DEFERRABLE
);
```

En esta tabla se definen los atributos `no_w_id, no_d_id` y `no_o_id` como llave primaria lo que impedirá que se inserten dos filas con esos tres campos iguales.

Con los atributos declarados como llave primaria postgreSQL crea automáticamente un índice para acelerar las búsquedas en esta tabla.

Se definen, además, una llave foránea compuesta por los atributos `no_w_id, no_d_id, no_o_id` referenciando a `o_w_id, o_d_id, o_id`. Esto impedirá que

se inserten filas en la tabla `orderr` cuyos campos de llave foránea no aparezcan en los campos referenciados.

Para la población inicial de la tabla `new_order` se utiliza la función `poblar_new_order()` y se insertan 900 filas por cada distrito, por lo tanto 9000 filas por cada almacén seleccionado por el usuario.

Durante el test, la cardinalidad de esta tabla no es fija. Por cada transacción New-Order ejecutada se inserta una nueva fila, mientras que por cada transacción Delivery se podrán eliminar hasta 10 filas.

Los métodos para la población inicial de esta tabla, usados para cumplir los requisitos de la cláusula 4.3 del TPC-C, son:

- `no_o_id` se le asigna el número de orden al que pertenece.
- `no_d_id` se le asigna el valor del distrito al que pertenece la orden.
- `no_w_id` se les asigna el valor del almacén al que pertenece la orden.

2.3.1.3 Generación de números aleatorios

Las funciones encargadas de obtener los datos aleatorios definidos por el estándar TPC-C están basadas en la función de C `random()`, que genera un número aleatorio de tipo long comprendido entre 0 y la constante del sistema `RAND_MAX`.

Para cumplir que cada variable aleatoria tenga una distribución probabilística determinada, es necesario que `random()` devuelva números aleatorios con distribución uniforme. Para ello, a cada variable aleatoria se le asigna un *vector de estado* que se encarga de procurar que los valores de `random()` para ese estado estén uniformente distribuidos.

La función `initstate()` crea un vector de estado de N posiciones a partir de una *semilla* (número entero) que se le pasa como parámetro. La llamada a la función devuelve el puntero al vector de estado.

Cada vez que se quiera obtener el siguiente valor para una variable, es necesario recuperar el estado asociado a ella. Esto se realiza mediante la función `setstate()` cuyo parámetro de entrada es el puntero al vector de estado. De esta manera la función `random()` generará para cada variable una secuencia de números distribuidos uniformemente.

Por lo tanto, a cada una de las variables aleatorias que han de tener una distribución uniforme se les asigna un único vector de estado. A aquellas variables en las que no es necesaria una distribución uniforme se les asigna un vector de estado común.

2.3.1.4 Tratamiento de señales

Cada uno de los módulos que componen el benchmark implementado, utilizan señales para controlar su ejecución.

Para que un proceso pueda capturar una determinada señal, es necesario hacer una llamada al sistema mediante la función `signal()` [16], que toma como parámetros el identificador de la señal y el puntero a la función para su tratamiento.

De ahora en adelante se denominará al proceso anterior como *enmascaramiento* de una señal.

Si se desea que un programa realice una determinada operación cuando se reciba uno de los tipos de señales definidos en UNIX, es necesario enmascarar esa señal con la función que realiza dicha operación.

A partir del momento en que el programa tenga la señal enmascarada, cada vez que la reciba se llamará automáticamente a su función de tratamiento.

En los módulos que implementan el benchmark algunas señales están enmascaradas con funciones de tratamiento que no realizan ninguna operación. Por ejemplo el Monitor de Transacciones y el Emulador de Terminal Remoto, enmascaran la señal `SIGINT` con funcione vacías, para que su ejecución no se vea afectada cuando el usuario teclea <ctrl-c>para parar la ejecución del test.

2.3.1.5 Comunicaciones ETR - MT

Como se ilustra la figura 2.3 del apartado de Consideraciones de Diseño, las comunicaciones entre los Emuladores de Terminal Remoto y el Monitor de Transacciones, se implementan a través de una cola de mensajes, memorias compartidas y semáforos de sincronización [16]. En este apartado se describen las estructuras de datos utilizadas para implementar las comunicaciones entre la población de ETRs y el MT. También se describen los mecanismos que implementan el envío de transacciones y la respuesta de los resultados, así como la forma en que el Monitor de Transacciones es capaz de identificar a cada uno de los Emuladores de Terminal Remoto.

Estructuras de datos para la comunicación

La estructura de mensaje que los ETR envían al MT, está definida en el fichero de cabecera tpcc.h y es la siguiente:

```
struct mensaje{
     long tipo;
     int id;
```

```
     union ttran{
          struct tmsgtrm msgtrm;
          struct tnew_order_men new_order;
          struct torder_status_men ostatus;
          struct tpayment_men payment;
          struct tdelivery_men delivery;
          struct tstock_level_men stock_level;
     } tran;
};
```

Esta estructura se compone de los siguientes campos:

- `tipo`: Identifica el tipo de mensaje enviado por el ETR. Los tipos disponibles son:
  - `MSGTRM`. El mensaje del terminal no se corresponde con una transacción sino representa una solicitud de conexión o desconexión con el MT.
  - `NEW_ORDER`. El mensaje contiene la solicitud de ejecución de la transacción New-Order.
  - `PAYMENT`. El mensaje contiene la solicitud de ejecución de la transacción Payment.
  - `ORDER_STATUS`. El mensaje contiene la solicitud de ejecución de la transacción Order-Status.
  - `DELIVERY`. El mensaje contiene la solicitud de ejecución de la transacción Delivery.
  - `STOCK_LEVEL`. El mensaje contiene la solicitud de ejecución de la transacción Stock_Level.

  Estos tipos de mensajes se definen como constantes.
- `id`: Contiene el identificador del ETR que envía el mensaje.
- `tran`: Este campo es una `union` donde se contienen los datos para cada tipo de mensaje. El acceso a este campo se determina a través del identificador de mensaje.

El formato de mensaje de respuesta que el MT escribe en la Memoria compartida, está definida en el fichero de cabecera tpcc.h, y es:

```
union tshm{
     int id;
     struct tnew_order new_order;
```

```
    struct tpayment payment;
    struct torder_status ostatus;
    struct tstock_level stock_level;
};
```

Esta `union` se compone de los siguientes campos:

- `id`: Identificador que el MT asigna al ETR en la etapa de conexión que se explicará más adelante.
- `new_order`: Estructura que contiene los datos de respuesta de la ejecución de la transacción New-Order.
- `payment`: Estructura que contiene los datos de respuesta de la ejecución de la transacción Payment.
- `ostatus`: Estructura que contiene los datos de respuesta de la ejecución de la transacción Order-Status.
- `stock_level`: Estructura que contiene los datos de respuesta de la ejecución de la transacción Stock-Level.

Creación de los canales de comunicación

- Cola de mensajes

El Monitor de Transacciones crea la cola de mensajes al iniciar su ejecución. La llamada al sistema para realizar esta operación es `msgget()` [16], toma como parámetros una llave de tipo `key_t` que se utiliza para determinar la cola a crear y una serie de *flags* utilizados para definir los permisos de acceso a la cola. Tras la llamada al sistema se obtiene un identificador con el que a partir de ese momento se accederá a la cola.

Para que un ETR pueda enviar un mensaje a través de la cola debe engancharse a ella utilizando la misma llave con la que la creó el MT. El ETR también se engancha utilizando la función `msgget()`.

La llave utilizada para crear la cola de mensajes está definida en el fichero de cabecera `tpcc.h`.

- Memoria compartida y semáforos de sincronismo.

Cada ETR crea al iniciar su ejecución un espacio compartido de memoria y un semáforo de sincronismo. La llave utilizada para realizar cada una de las operaciones anteriores se pasa como parámetro al módulo cuando es invocado.

La memoria compartida se reserva mediante la función `shmget()` [16], que tiene como parámetros la llave de la memoria compartida y los *flags* que determinan los permisos de la memoria. Esta función devuelve el identificador de la memoria. Para que el ETR pueda realizar operaciones sobre la memoria que ha creado debe obtener un puntero a ésta. Esto se realiza mediante la función `shmat()` [16], que toma como entrada el identificador del la memoria y devuelve un puntero con la posición de la memoria compartida dentro del espacio virtual de direcciones del proceso.

El semáforo de sincronismo se obtiene mediante la función `semget()`[16], utilizando la misma llave y los mismos permisos que en la creación de la memoria compartida. Esta función devuelve el identificador del semáforo gracias al cual se podrán realizar operaciones en él.

Si el valor de la llave que se ha pasado como parámetro al ETR identifica a una memoria o un semáforo ya existentes, el ETR busca otro valor de la llave para crear la memoria compartida o el semáforo. De esta forma todos los ETR que intervienen en el test tienen memorias y semáforos distintos.

Etapas de la comunicación

- Etapa de Conexión

Cuando el ETR ha creado la memoria compartida junto con el semáforo de sincronismo y el MT la cola de mensajes, para cerrar el canal de comunicaciones, es necesario que el MT se enganche a cada una de las memorias compartidas de cada uno de los ETR y que conozca la dirección cada semáforo asociado.

Para ello en esta etapa el ETR envía través de la cola un mensaje de tipo MSGTRM (mensaje de terminal) con los identificadores de la memoria compartida y del semáforo. A este mensaje se le llama *Mensaje de Conexión.*

La estructura utilizada para acceder al campo `tran` del mensaje de tipo MSGTRM, está definida en el fichero de cabecera `tpcc.h`, y es la siguiente:

```
struct tmsgtrm {
     int codctl;
     key_t sem_llave;
     key_t shm_llave;
}
```

Esta estructura se compone de los siguientes campos:

- `codctl`: determina si el mensaje es una solicitud de conexión o de desconexión.

- `sem_llave`: contiene la llave del semáforo de sincronismo.
- `shm_llave`: contiene la llave de la memoria compartida.

Cuando el MT recibe el mensaje, para engancharse a la memoria compartida primero debe obtener un identificador por medio de la función `shmget()` utilizando la llave recibida y obtener el puntero a la memoria mediante la función `shmat()`. Para obtener el identificador del semáforo debe llamar a la función `semget()`, utilizando la llave recibida.

El MT almacena los identificador de semáforo y de memoria compartida y el puntero a esta, en la primera posición libre de un vector de estructuras con el formato siguiente:

```
struct tclientes{
     int shmid;
     int semid;
     union tshm *shm;
}
```

donde:

- `shmid`: es el identificador de la memoria compartida.
- `semid`: es el identificador del semáforo.
- `shm`: apunta a la memoria compartida.

Esta estructura está definidas en el módulo MT. El tamaño del vector de estructuras está definido por una constante declarada en el MT, y que tiene el valor de 1000.

La posición libre encontrada se escribe en el campo `id` de la memoria compartida y constituirá el identificador del terminal. El ETR lo utilizará para que el MT pueda identificarle.

De esta forma el ETR ya está perfectamente identificado de cara al MT. Cuando este último reciba en mensaje de transacción procedente de un ETR, con sólo mirar su identificador (campo `id` de la estructura de mensaje) podrá localizar en el vector anterior el puntero a la memoria y el identificador de semáforo correspondientes, pudiendo así contestar al ETR.

Envío de Transacciones y Recepción de Resultados.

Cuando un ETR genera los datos de una transacción seleccionada, los envía a través de la cola mediante la correspondiente estructura de mensaje. El envío de un mensaje se realiza mediante la llamada al sistema `msgsnd()` [16] que recibe como parámetros la estructura de datos del mensaje y el identificador de la cola.

Las estructuras correspondientes al campo `tran` de la estructura de mensaje para cada tipo de transacción están definidas en el fichero de cabecera `tpcc.h` y son las siguientes:

Transacción New-Order:

```
struct tnew_order_men {
     int w_id;
     int d_id;
     long c_id;
     struct articulo {
          long ol_i_id;
          long ol_supply_w_id;
          int ol_quantity;
          char flag;
     } item[15];
}
```

El significado de cada campo es:

- `w_id` número de almacén del ETR.
- `d_id` distrito seleccionado.
- `c_id` identificador de cliente seleccionado.
- vector `item` representa cada línea de orden de la transacción New-Order. Se compone de los siguientes campos:
  - `ol_i_id` identificador de articulo.
  - `ol_supply_w_id` identificador del almacén de suministro.
  - `ol_quantity` cantidad del articulo.
  - `flag` indica si la posición del vector contiene un artículo o está vacío.

Transación Payment:

```
struct tpayment_men {
     long w_id;
```

```
    long d_id;
    long c_id;
    long c_w_id;
    long c_d_id;
    char c_last[16];
    double h_amount;
}
```

El significado de cada campo es:

- `w_id` número de almacén del ETR.
- `d_id` distrito seleccionado.
- `c_id` identificador de cliente seleccionado.
- `c_w_id` identificador del almacén al que pertenece el cliente.
- `c_d_id` identificador del distrito al que pertenece el artículo.
- `c_last` contiene el nombre del cliente.
- `h_amount` cantidad del pago a registrar.

Transacción Order-Status:

```
struct torder_status_men {
    long w_id;
    long d_id;
    long c_id;
    char c_last[16];
}
```

El significado de cada campo es:

- `w_id` número de almacén del ETR.
- `d_id` distrito seleccionado.
- `c_id` identificador de cliente seleccionado.
- `c_last` contiene el nombre del cliente.

Transacción Stock-Level:

```
struct tstock_level_men {
     long w_id;
     long d_id;
     long threshold;
}


El significado de cada campo es:

\begin{itemize}
\item \texttt{w\_id} número de almacén del ETR.
\item \texttt{d\_id} número de distrito del ETR.
\item \texttt{threshold} valor umbral.
\end{itemize}

Transacción Delivery:


\begin{verbatim}
struct tdelivery_men {
     time_t seg;
     unsigned short mseg;
     long w_id;
     long d_id;
     int o_carrier_id;
};
```

El significado de cada campo es:

- `seg` segundos del sello de hora tomado antes de el encolado de la transacción.
- `mseg` milisegundos del sello de hora tomado antes del encolado de la transacción.
- `w_id` número de almacén del ETR.
- `d_id` número de distrito del ETR.
- `o_carrier_id` identificador del transportista.

Tras el envío del mensaje el ETR intenta restar una unidad al valor del semáforo. Como el semáforo no puede tomar un valor negativo y su valor es cero al iniciar la comunicación, el resultado es que el ETR espera a que el semáforo tenga un valor positivo. Cuando el MT recibe y ejecuta la transacción fuerza a uno el valor del semáforo, permitiendo que el ETR acceda a los datos de respuesta.

Como consecuencia el semáforo vuelve a tomar el valor 0 y queda preparado para la siguiente comunicación.

Las modificaciones del valor del semáforo se realizan a través de la función `semop()` [16] que toma como parámetros el identificador del semáforo y una estructura de datos que contiene la operación a realizar.

En el caso de la transacción Delivery el ETR y el MT no efectúan ninguna modificación en el semáforo, ya que la ejecución de esta transacción es aplazada (como se especifica en la cláusula 2.7.2 del TPC-C) y yo necesita de una respuesta por parte del MT. El ETR se limita a enviar la transacción y pasa a la siguiente.

Las estructuras de datos en las que el MT escribe los resultados de la transacción contienen los datos que los ETR mostrarán en las pantallas de Entrada/Salida, según se especifica en los perfiles de transacción de la cláusula 2 del TPC-C. Para cada transacción, las estructuras de mensajes de respuesta están definidas en el fichero de cabecera `tpcc.h`, y son:

Transacción New-Order:

```
struct tnew_order {
     char c_last[17];
     char c_credit[3];
     char o_entry\_d[21];
     float c_discount;
     long o_id;
     int o_ol_cnt;
     float w_tax;
     float d_tax;
     char ctl;
     float total_amount;
     struct res_artic {
          char i_name[25];
          int s_quantity;
          char b_g;
          float i_price;
          float ol_amount;
     } item[15];
};
```

El significado de cada campo es:

- `c_last` nombre del cliente seleccionado.
- `c_credit` estado del crédito del cliente.

- `o_entry_d` fecha de la orden.
- `c_discount` tasa de descuento del cliente.
- `o_id` identificador de la orden.
- `o_ol_cnt` número de líneas que componen la orden.
- `w_tax` tasa del almacén.
- `d_tax` tasa del distrito.
- `ctl` indica si la transacción ha sido confirmada o rechazada.
- `total_amount` suma total de la orden.
- vector `item` representa los datos de respuesta por cada linea de orden de la transacción. Se compone de los siguientes campos:
  - `i_name` contiene el nombre del artículo.
  - `s_quantity` contiene el stock del artículo.
  - `b_g` contiene el distintivo `B` o `G` del artículo.
  - `i_price` contiene el precio del artículo.
  - `ol_amount` contiene el precio total de la línea de orden.

Transacción Payment:

```
struct tpayment {
     char w_street_1[21];
     char w_street_2[21];
     char w_city[21];
     char w_state[3];
     char w_zip[10];
     char d_street_1[21];
     char d_street_2[21];
     char d_city[21];
     char d_state[3];
     char d_zip[10];
     long c_id;
     char c_first[17];
     char c_middle[3];
     char c_last[17];
     char c_street_1[21];
     char c_street_2[21];
     char c_city[21];
     char c_state[3];
```

```
    char c_zip[10];
    char c_phone[17];
    char c_credit[3];
    char c_since[21];
    float c_credit_lim;
    float c_discount;
    float c_balance;
    char c_data[501];
    char h_date[21];
}
```

El significado de cada campo es:

- `w_street_1, w_street_1` son las calles del almacén.
- `w_city` es la ciudad del almacén.
- `w_state` es el estado del almacén.
- `w_zip` es el código postal del almacén.
- `d_street_1, d_street_2` son las calles del distrito.
- `d_city` es la ciudad del distrito.
- `d_state` es el estado del distrito.
- `d_zip` es el código postal del distrito.
- `c_id` es el identificador del cliente.
- `c_first, c_middle, c_last` constituye el nombre completo del cliente.
- `c_street_1, c_street_2` son las calles del cliente.
- `c_city` es la ciudad del cliente.
- `c_state` es el estado del cliente.
- `c_zip` es el código postal del cliente.
- `c_phone` es el número de teléfono del cliente.
- `c_credit` es el crédito del cliente.
- `c_since` es una fecha asociada al cliente.
- `c_credit_lim` es el límite de crédito del cliente.
- `c_discount` es la tasa de descuento del cliente.

- `c_balance` es el balance del cliente.
- `c_data` es información a cerca del cliente
- `h_date` es la fecha de entrada de la transacción.

Transacción Order-Staus:

```
struct torder_status {
     long c_id;
     char c_first[17];
     char c_middle[3];
     char c_last[17];
     float c_balance;
     long o_id;
     char o_entry_d[21];
     long o_carrier_id;
     int num_art;
     struct res_artic_o_s {
          long ol_supply_w_id;
          long ol_i_id;
          int ol_quantity;
          float ol_amount;
          char ol_delivery_d[21];
     } item[15];
};
```

El significado de cada campo es:

- `c_id` nombre el identificador del cliente seleccionado.
- `c_first` `c_middle` y `c_last` determinan el nombre completo del cliente.
- `c_balance` es el balance del cliente.
- `o_id` identificador de la orden analizada.
- `o_ol_cnt` número de líneas que componen la orden.
- `o_entry_d` fecha de la orden.
- `o_cariier_id` es el número de transportista.
- `num_art` indica el número de artículos de que se componía la orden.
- vector `item` representa los datos de respuesta de cada línea de orden de la transacción analizada. Se compone de los siguientes campos:

- `ol_supply_w_id` identifica al almacén al que pertenece el artículo.
- `ol_i_id` contiene el identificador de artículo.
- `ol_quantity` cantidad del artículo.
- `ol_delivery_d` contiene la fecha de reparto de la orden.

Transacción Stock-Level:

```
struct tstock_level {
     int lowstock;
}
```

- `lowstock` contiene el número de artículos cuyo stock está por debajo del umbral.

Etapa de desconexión

Una vez terminado el periodo de medida, el Controlador del Benchmark envía a todos los ETR una señal indicándolo. En ese momento cada ETR envía un mensaje avisando al MT que debe desengancharse de la memoria compartida para poder eliminarla correctamente y espera a que este ultimo abra el semáforo. A este mensaje se le llama *Mensaje de Desconexión*. La estructura de este mensaje es idéntica a la del Mensaje de Conexión. Unicamente se tiene en cuenta el campo `codctl` que indica que es un mensaje de desconexión.

El MT se desenganchará de la memoria compartida mediante las función `shmdt()` [16] que recibe como parámetro el puntero a la memoria.

Tras desengancharse, abre el semáforo permitiendo al ETR desengancharse también mediante `shmdt()`,

Para eliminar la memoria compartida y el semáforo de sincronismo, el ETR utiliza las funciones `shmctl()` [16] y `semctl()` [16] respectivamente.

Cuando todos los ETR terminan su ejecución, el Controlador del Benchmark envía una señal al MT indicándole que el test ha finalizado. Es ese momento el MT borrará la cola de mensajes mediante la función `msgctl()` [16]. De esta forma no quedará ningún mensaje pendiente en la cola cuando el MT la elimine.

2.3.1.6 Ficheros de salida del benchmark

A continuación se describe el formato de los ficheros que genera el benchmark.

Bitácora del ETR.

En este fichero se almacenan los datos de las transacciones que se ejecutan, necesarios parar el posterior cálculo del rendimiento. Por cada transacción realizada, el ETR inserta una nueva línea de datos. Los ETR escriben este fichero en el directorio `/tmp`, y tras finalizar el test lo mueven al directorio `/usr/share/var/tpcc`.

Como se dijo en la introducción de esta memoria, este benchmark se va a utilizar en el futuro para medir el rendimiento de un cluster basado en memoria compartida distribuida. El árbol de directorios del cluster es común a todos los nodos, salvo el directorio `/tmp`. A fin de no sobrecargar la red de comunicaciones del cluster durante el test, la escritura de las bitácoras se realiza en ese directorio. Fuera del periodo de test, se mueven a un directorio compartido para poder realizar el recuento de resultados.

El nombre del fichero de cada ETR se determina a partir de los números de almacén y distrito al que pertenece. El formato del nombre del fichero es el siguiente:

`F_W_D.log`

donde `W` es el número de almacén, y `D` el número de distrito.

El formato de cada una de las líneas que se insertan en la bitácora son:

**it> s1 m1 s2 m2 tr tp f1 f2 f3 s3 m3*

donde:

- *it* es el identificador de tipo de transacción.
- *s1* son los segundos del sello de hora del inicio de transacción.
- *m1* son los milisegundos del sello de hora del inicio de transacción.
- *s2* son los segundos del sello de hora previo al encolado de transacción.
- *m2* son los milisegundos del sello de hora previo al encolado de la transacción
- *tr* es el tiempo de respuesta de la transacción.
- *tp* es el tiempo de pensar que se ha esperado en esa transacción.
- *f1, f2* y *f3* tienen significados que dependen del tipo de transacción.
- *s3* son los segundos del sello de hora del la finalización de la transacción.
- *m3* son los milisegundos del sello de hora del la finalización de la transacción.

Bitácoras del MT.

EL Monitor de Transacciones escribe dos bitácoras de resultados, `tm_delivery_res.log` y `tm_delivery_tr.log`, y otra de errores de transacción `tm_err.log`. Estas bitácoras se escriben en el directorio `/usr/share/var/tpcc`.

Bitácora `tm_delivery_res.loq`.

Contiene los resultados de la ejecución de las transacciones Delivery, según se especifica en el perfil de transacción de la cláusula 2.7 del estándar TPC-C . Para cada transacción se escriben una serie de líneas que contienen los resultados. El formato de los resultados de cada transacción es:

`Inicio Transaccion. Fecha Encolado:` *s1 m1*`, Almacén` *W*`, Repartidor` *R*

para cada uno de los 10 distritos del almacén donde se ejecuta la orden se escribe:

ch d num

y se finaliza con:

`Terminada a las:` *s2 m2*

donde:

- *s1* son los segundos del sello de hora enviado por el cliente.
- *s2* son los milisegundos del sello de hora enviado por el cliente.
- *W* almacén al que pertenece el cliente.
- *R* es el numero de repartidor seleccionado.
- *ch* es el carácter `'#'` o `'*'`, que indica si la orden ha sido correctamente repartida o si se ha saltado el distrito por que no tenía ninguna orden pendiente, respectivamente.
- *d* es el distrito donde se reparte la orden.
- *num* es el identificador de la orden repartida.
- *s2* son los segundos del sello de hora de finalización de la transacción.
- *m2* son los milisegundos del sello de hora de finalización de la transacción.

Bitácora `tm_delivery_tr.loq`.

En este archivo se registran los tiempos de ejecución de la transacción Delivery, para el posterior cálculo del rendimiento.

Para cada transacción Delivery ejecutada se inserta un nueva línea con el formato:

W D s1 m1 s2 m2

donde:

- *W* es el almacén al que pertenece el terminal que envió la transacción.
- *D* es el distrito al que pertenece el terminal que envió la transacción.
- *s1* son los segundos del sello de hora del inicio de la transacción enviado por el ETR
- *m1* son los milisegundos segundos del sello de hora del inicio de la transacción enviado por el ETR
- *s2* son los segundos del sello de hora de la finalización de la transacción.
- *m2* son los milisegundos del sello de hora de la finalización de la transacción.

Bitácora `tm_err.log`.

En este fichero se registra información acerca de los posibles errores producidos en la ejecución de las transacciones. No tiene formato explícito. Se muestra el tipo de transacción que se estaba ejecutando y una breve descripción del error producido.

Bitácora de limpiezas

El Controlador de Limpiezas escribe una entrada en este fichero por cada limpieza realizada sobre la base de datos. Se encuentra en el directorio `/usr/share/var/tpcc/` y su nombre es `vacuum.log`. El formato es el siguiente:

`1° Sello` *s1 m1* `2° Sello` *s2 m2*

donde:

- *s1* son los segundos del sello de hora al comienzo del vacuum.
- *s2* son los milisegundos del sello de hora al comienzo del vacuum.
- *s2* son los segundos del sello de hora al fin de la limpieza.
- *m2* son los milisegundos del sello de hora al fin del vacuum.

Bitácora de checkpoints

El Controlador de Checkpoints escribe una entrada en este fichero por cada chekpoint realizado sobre la base de datos. Se encuentra en el directorio `/usr/share/var/tpcc/` y su nombre es `check.dat`. El formato es el siguiente:

`1° Sello` *s1 m1* `2° Sello` *s2 m2*

donde:

- *s1* son los segundos del sello de hora al comienzo del checkpoint.
- *s2* son los milisegundos del sello de hora al comienzo del checkpoint.
- *s2* son los segundos del sello de hora al fin del checkpoint.
- *m2* son los milisegundos del sello de hora al fin del checkpoint.

Fichero de constantes

En este fichero se almacenan las constantes aleatorias utilizadas en el proceso de carga de la base de datos. Los clientes necesitan este fichero para generar los campos C_LAST, C_ID y OL_I_ID, según se especifica en la cláusula 2.1.6 del TPC-C. Se encuentra en el directorio `/usr/share/var/tpcc` y su nombre es `cons.dat`.

El formato del fichero es el siguiente:

A B C
A' B' C'

donde:

- *A* y *A'* son el valor de la constante C_LOAD (constante utilizada en la carga de la base de datos) y C_RUN (constante utilizada para la generacion de datos de transacción)para el campo C_LAST.
- *B* y *B'* son el valor de la constante C_LOAD y C_RUN para el campo C_ID.
- *C* y *C'* son el valor de la constante C_LOAD y C_RUN para el campo OL_I_ID.

Las constantes C_LOAD y C_RUN se calculan en el proceso de carga de la base de datos, cumpliendo con que el valor absoluto de la diferencia entre ambos

sea un valor perteneciente a [95,119], excluyendo los valores 65 y 112, según se especifica en la cláusula 2.1.6.

Fichero de resultado de limpiezas

En el recuento de resultados se genera este fichero que contiene información acerca de las limpiezas efectuadas durante el Test. Se encuentra en el directorio donde se ha invocado al Controlador del Benchmark y se ha realizado el recuento. Su nombre es `vacuum.dat` y el formato es el siguiente:

`Hora de Comienzo`	`Tiempo desde el Inicio (s)`	`Duración (s)`
Fecha	*Transcurrido*	*Duración*
.	.	.
.	.	.
.	.	.

donde

- *Fecha*: es la fecha y hora del comienzo de la limpieza.
- *Transcurrido*: es el tiempo transcurrido desde el inicio del test.
- *Duración*: es la duración de la limpieza.

Fichero de resultado de checkpoints.

En el recuento de resultados se genera este fichero que contiene información acerca de los checkpoints efectuados durante el Test. Se encuentra en el directorio donde se ha invocado al Controlador del Benchmark y se ha realizado el recuento. Su nombre es `check.dat`. El formato del fichero es el siguiente:

`Hora de Comienzo`	`Tiempo desde el Inicio` (s)	`Duración (s)`
Fecha	*Transcurrido*	*Duración*
.	.	.
.	.	.
.	.	.

donde

- *Fecha*: es la fecha y hora del comienzo del checpoint.
- *Transcurrido*: es el tiempo transcurrido desde el inicio del test.
- *Duración*: es la duración de la checkpoint.

Ficheros de puntos

Estos ficheros contienen los puntos generados en el recuento de resultados del test. Son necesarios para la visualización de las gráficas que se especifican en la cláusula 5.6 del estándar TPC-C. Se encuentran en el directorio donde se ha llamado al Controlador del Benchmark y se ha ejecutado el recuento.

- Los ficheros de la gráfica 1 definida en la cláusula 5.6.1 son:
  - New-Order: `g1NewOrder.dat`
  - Payment: `g1Payment.dat`
  - Order-Status: `g1OrderStatus.dat`
  - Delivery: `g1Delivery.dat`
  - Stock-Level: `g1Stock-Level.dat`
- El fichero de la gráfica 3 definida en la cláusula 5.6.3 es `g3.dat`.
- El fichero de la gráfica 4 definida en la cláusula 5.6.3 es `g4.dat`.

Estos ficheros se componen de pares de puntos (x, y) con el siguiente formato:

```
x1 y1
x2 y1
. .
. .
xn yn
```

Fichero `medida.log`

El fichero `medida.log` lo genera el Controlador del Benchmark cuando realiza un test, y contiene información sobre éste. Su formato es el siguiente:

```
TEST REALIZADO EL fecha.
Número de almacenes: w.
Número de terminales por almacen: d .
Intervalo de medida: med minutos.
Intervalo de rampa: ramp minutos.
Intervalo entre vacuums: int.
Número máximo de vacuums: num .
Sello de hora del comienzo del test: seg1 mseg1 .
Sello de hora del comienzo del intervalo de medida: seg2 mseg2 .
Sello de hora de finalización de intervalo de medida: seg3 mseg3 .
Sello de hora de finalización del test: seg4 mseg4 .
```

donde:

- *fecha* es la fecha y la hora en la que comenzó el test.
- *w* y *d* son el número de almacenes y de terminales por almacén con que se ejecuto el test respectivamente.
- *med* y *ramp* son los intervalos de medida y de rampa del test respectivamente
- *int* es el intervalo entre limpiezas de la base de datos. Si no se han realizado su valor es 0.
- *num* es al número máximo de limpiezas realizadas. Si realizaron limpiezas hasta la finalización del test, su valor es 0.
- *seg1 mseg1* son los segundos y milisegundos del sello de hora del comienzo del test.
- *seg2 mseg2* son los segundos y milisegundos del sello de hora del comienzo del intervalo de medida.
- *seg3 mseg3* son los segundos y milisegundos del sello de hora del finalización del intervalo de medida.
- *seg4 mseg4* son los segundos y milisegundos del sello de hora del finalización del test.

Fichero de resultados del test

Este fichero se genera durante el recuento. Está situado en el mismo directorio donde se ha invocado al Controlador del Benchmark y se ha hecho el recuento. El usuario decide si desea crear este fichero, así como su nombre. Contiene los resultados del test de rendimiento y su formato es el mismo que el de los resultados mostrados por pantalla tras realizar un recuento.

2.3.2. Controlador del Benchmark

2.3.2.1 Esquema general

El controlador del benchmark es el módulo principal de todo el sistema. Se encarga de proporcionar al usuario la interfaz que le permite seleccionar las operaciones disponibles con el fin de hacer una evaluación completa del rendimiento de la máquina. Además se encarga de poner en funcionamiento todos los mecanismos y de efectuar todos los controles necesarios para realización de dichas operaciones.

Debido a que el controlador del benchmark ejecuta procesos concurrentes, se utilizarán diagramas de flujo de datos para explicar su implementación, a pesar de que este tipo de diagramas estén más relacionados con la etapa de diseño.

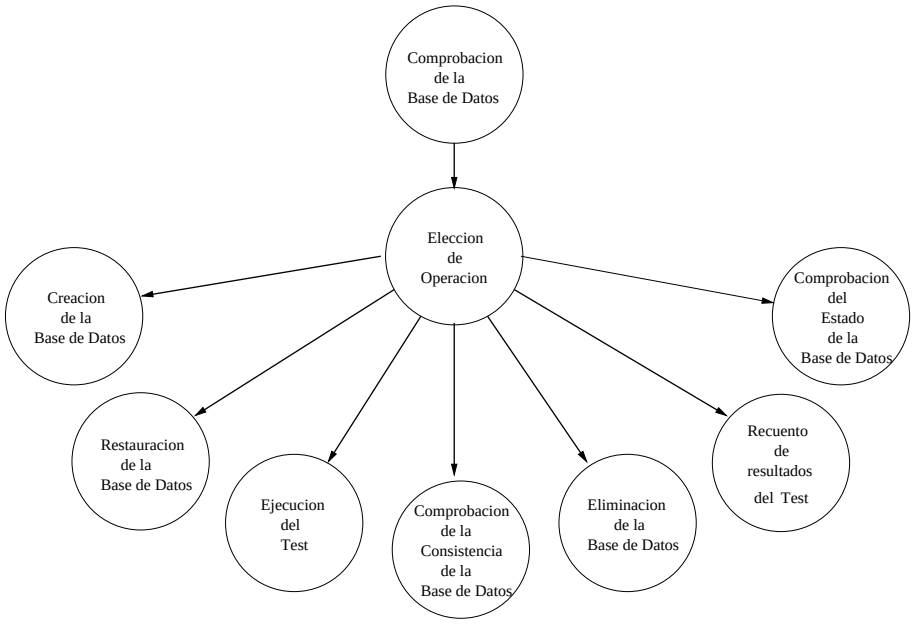


Figura 2.5: Esquema general del Controlador del Benchmark

El módulo Controlador del Benchmark se compone de las diferentes etapas representadas en el diagrama de flujo de datos de la figura 2.5.

2.3.2.2 Etapa de Comprobación de la Existencia de la Base de Datos

Es la primera etapa que se ejecuta. Comprueba la existencia de la base de datos `tpcc` utilizada para el test. La comprobación se realiza conectando con postgreSQL a través de la base de datos `template1` que postgreSQL instala por defecto. La conexión con está base de datos permitirá crear y eliminar la base de datos `tpcc`. El resultado de la comprobación se refleja en un flag llamado `haybd` que indicará en lo sucesivo la existencia de la base de datos `tpcc`. Este flag tomará el valor '1' en caso de que exista la base de datos y '0' en caso contrario.

El la etapa de comprobación se compone de las siguientes etapas:

1. Conexión con `template1`.
2. Activación de modo *autocommit* en la conexión a `template1`.
3. Comprobación de la existencia de `tpcc`.

   La existencia de la base de datos se comprueba forzando la creación de una base de datos `tpcc`. Si la base de datos ya existía postgreSQL la estuctura de datos `sqlca` indicará que se ha producido un error.

   En primer lugar se crea la base de datos `tpcc` mediante la sentencia:

```
EXEC SQL CREATE DATABASE tpcc;
```

Si tras esto no se ha producido ningún error se borrará la base de datos tpcc, para que sea el usuario quien posteriormente decida su creación. Además se indicará la no existencia de `tpcc` fijando el valor de `haybd` a 0. El borrado de `tpcc` se realiza mediante la siguiente sentencia:

```
EXEC SQL DROP DATABASE tpcc;
```

En caso de que se haya producido error, si este es el '-400' indicará que `tpcc` existía previamente, y se fija el valor de `haybd` a 1. En caso de que se haya producido cualquier otro error el programa termina anormalmente, informando al usuario mediante un mensaje en la pantalla con el número y de la descripción del error.

4. Desconexión de `template1`.

2.3.2.3 Etapa de Selección de Operación

A la etapa de comprobación le sucede la etapa de selección de operación que solicita al usuario la operación a realizar. Esta etapa toma como entrada el flag `haybd` que indica la existencia de la base de datos para el test. La finalidad de este flag se indicará posteriormente. Las salida es una acción de control sobre el la etapa que controla la operación solicitada por el usuario.

La etapa de elección se compone de los procesos indicados en la figura 2.6.

Etapa Comprobación de Bitácoras. Detecta si se ha realizado un test con anterioridad, comprobando la existencia del archivo medida.log. Este archivo se genera tras la ejecución de un test y su información es necesaria para el cálculo del rendimiento ofrecido. La existencia o no se indica en un flag llamado `haylogs`. Si `haylogs` es '1' significa que se ha realizado un test, y por el contrario si es '0' indica que no se ha ejecutado ningún test. La finalidad de este flag se explica posteriormente.

Etapa menú. La Etapa Menú proporciona la interfaz con el usuario permitiendo la elección de la operación a realizar. Toma como entrada los flags anteriormente mencionados `haybd` y `haylogs`. Estos flags determinan las operaciones permitidas en cada interacción con el usuario. La salida de la etapa es la opción seleccionada. La figura 2.7 representa la pantalla de menú.

Como se ha dicho la posibilidad de elegir una opción depende de los flags `haybd` y `haylogs`. En la pantalla de salida se tiene en cuenta esa consideración de forma que:

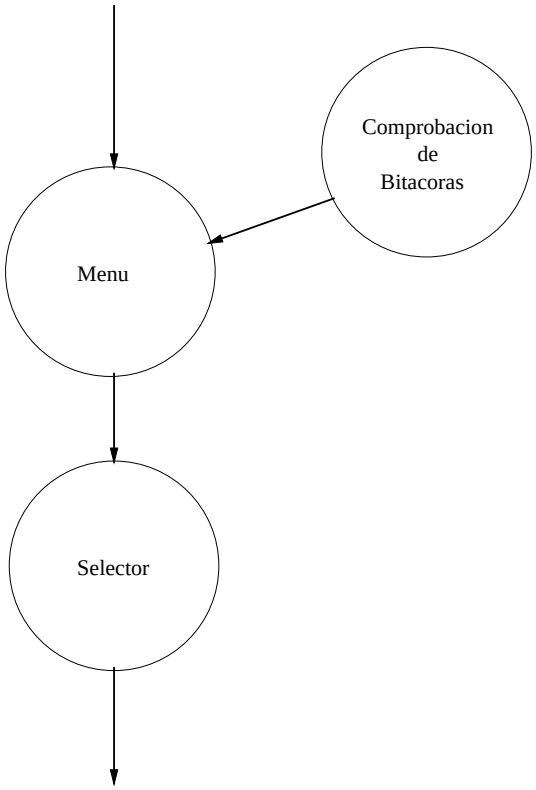


Figura 2.6: Etapa de selección de 0peración

```
        +-------------------------------------------------+
        | BENCHMARK TPCC  --- UNIVERSIDAD DE VALLADOLID --- |
        +-------------------------------------------------+


        1.- CREAR UNA NUEVA BASE DE DATOS PARA EL TEST.
        2.- RESTAURAR UNA BASE DE DATOS EXISTENTE.
        3.- EJECUTAR EL TEST.
        4.- COMPROBAR LA CONSISTENCIA DE LA BASE DE DATOS.
        5.- ELIMINAR LA BASE DE DATOS.
        6.- VER RESULTADOS DEL TEST.
        7.- COMPROBAR ESTADO DE LA BASE DE DATOS.


        8.- Salir.

SELECCIONE OPERACION:
```

Figura 2.7: Pantalla de menú

- Si `haybd` contiene el valor cero, sólo se muestra la opción '1' de creación de la base de datos para el test.

- Si `haybd` contiene el valor uno, se muestran las siguientes opciones:

  2.- Restaurar la base de datos.
  3.- Ejecutar el test.
  4.- Comprobar la consistencia de la base de datos.
  5.- Eliminar la base de datos.
  7.- Ver el estado de la base de datos.

- Si `haylogs` contiene el valor uno, se muestra la opción '6' para realizar el recuento del último test.

- La opción '8' para salir del programa se muestra independientemente de los flags.

Esta etapa se implementa a través de la función `menu()`.

Etapa selector. La etapa selector toma como entrada la operación seleccionada por el usuario a través de la etapa Menú y se encarga de activar la etapa que controla dicha operación. Si la operación solicitada es 'salir' (opción 8 en el menú), el proceso selector se encarga de poner los medios para la interrupción del programa.

2.3.2.4 Etapa Creación de la Base de Datos

Esta etapa se activa cuando el usuario selecciona la opción '1' del menú. Permite crear una base de datos preparada para la ejecución de un test de rendimiento con el tamaño elegido por el usuario. En la figura 2.8 se ilustran las operaciones de las que se compone, que son:

Etapa Petición de Datos. Se encarga de solicitar al usuario el número del almacenes de que constará la base de datos. Los datos introducidos se validan antes de producir una salida hacia los demás procesos. Para ser válido el número introducido debe ser mayor que 0 y menor que la constante `NUM_MAX_ALM` definida en el fichero de cabecera `tpcc.h`. La constante `NUM_MAX_ALM` determina el número máximo de almacenes. Está fijada en 100.

Tras haber obtenido un número válido de almacenes. La etapa de petición de datos activa la etapa Control de Carga.

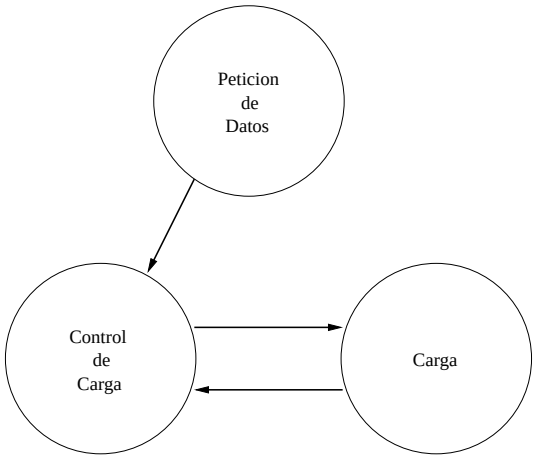


Figura 2.8: Etapa de Creación de la Base de Datos

Para dar paso a las dos etapas siguientes: Control de Carga y Carga, el programa se divide en dos procesos mediante la llamada de sistema `fork()` de UNIX [16]. El proceso hijo se encarga de ejecutar la etapa Carga y el padre de ejecutar la etapa Control de Carga.

Etapa Control de Carga. La etapa Control de Carga tiene la capacidad de activar, congelar o desactivar la etapa de carga. Tiene como entradas las señales UNIX `SIGINT` y `SIGUSR1`. La salida son las señales `SIGSTOP`, `SIGCONT` y `SIGTERM` que controlan la etapa de Carga.

La señal `SIGINT` se recibe cuando se pulsan simultáneamente las teclas 'control' y 'C' en el teclado <ctrl-C>. En este momento, se envía la señal `SIGSTOP` a la etapa de carga para congelar su ejecución y se solicita al usuario autorización para desactivar la etapa de carga. Si el usuario decide continuar con la carga se envía la señal `SIGCONT` a la etapa de carga para reactivar su ejecución. En caso contrario se envía la señal `SIGTERM` que provoca la desactivación de la etapa de Carga y el fin de la ejecución de la etapa de Creación de la Base de Datos.

La señal `SIGINT` se enmascara con la función `sig_int1()` que se encarga de interactuar con el usuario para solicitar la interrupción de la carga.

La señal `SIGUSR` la envía la etapa de carga cuando ha terminado su ejecución. Cuando la etapa de Control de Carga la recibe, termina la etapa Creación de la Base de Datos. Esta señal se enmascara mediante la función `sig_usr1()`.

Etapa de Carga. La etapa de carga toma como entrada el número de almacenes determinado en la etapa de Petición de Datos y las señales `SIGSTOP`, `SIGCONT` y `SIGTERM` que provienen de la etapa de Contol de Carga. La salida resultante es

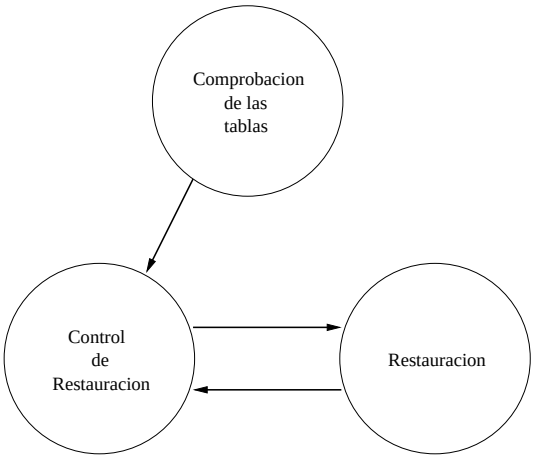


Figura 2.9: Etapa de Restauración de la Base de Datos

la base de datos `tpcc`, el fichero de constantes `cons.dat` que contiene las constantes aleatorias utilizadas durante el proceso de carga y la señal `SIGUSR1` que se envía a la etapa de Control de Carga.

En la etapa de carga se crea una nueva base de datos `tpcc` a través de postgreSQL, se crean las nueve tablas y se pueblan consecutivamente. Tanto la generación de los datos de los atributos, como la cardinalidad de las tablas, así como los requisitos para la integridad referencial se realizan siguiendo las especificaciones de las cláusulas 1 y 4 del estándar TPC-C. Las sentencias utilizadas para la creación de las tablas, así como los mecanismos de generación de los valores de cada uno de los atributos para la población inicial, se explican en el apartado 2.3.1.2.

A medida que se van insertando filas en las tablas se muestran mensajes por pantalla que informan de la evolución del proceso de carga.

Para la creación de la base de datos se utiliza la función `creartablas()`.

La función `SIGTERM` se enmascara mediante la función `sig_term_exit()` que se encarga de finalizar la etapa de carga.

2.3.2.5 Etapa de Restauración de la Base de Datos

Esta etapa se activa cuando el usuario selecciona la opción '2'. Permite eliminar las modificaciones producidas por las transacciones de un test anterior, obteniendo una base de datos preparada para el test. En la figura 2.9 se ilustran las operaciones de las que se compone:

Etapa Comprobación de las Tablas. Se encarga de examinar la base de datos para comprobar que la restauración se puede llevar a cabo. Se comprueba la existencia de todas las tablas de la base de datos así como el número de filas de las tablas cuya cardinalidad no sufre variaciones. La finalidad de esto es evitar realizar la restauración en una base de datos errónea en la que, por ejemplo, se haya eliminado alguna de las tablas, o que alguna se ellas no tenga la cardinalidad necesaria.

Si el resultado de la comprobación es erróneo, se finaliza con la ejecución de la etapa de Restauración de la base de datos. En caso contrario comienza la ejecución de las etapas Control de Restauración y Restauración. Para dar paso a estas dos etapas, el programa se divide en dos procesos mediante la llamada de sistema `fork()`. El proceso hijo se encarga de ejecutar la Etapa de Restauración y el padre del de la Etapa de Control de Restauración.

Etapa de Control de Restauración. La etapa Control de Restauración tiene la capacidad de activar, congelar o desactivar la etapa de Restauración. Tiene como entradas las señales UNIX `SIGINT` y `SIGUSR1`. La salida son las señales `SIGSTOP`, `SIGCONT` y `SIGTERM` que controlan la etapa de carga.

La señal `SIGINT` se recibe cuando se pulsan simultáneamente las teclas 'control' y 'C' en el teclado <ctrl-C>. En este momento, se envía la señal `SIGSTOP` a la etapa de restauración para congelar su ejecución y se solicita al usuario autorización para desactivarla. Si el usuario decide continuar con la restauración se envía la señal `SIGCONT` a la etapa de Restauración para reactivar su ejecución. En caso contrario se envía la señal `SIGTERM` que provoca la desactivación de la etapa de Restauración y el fin de la ejecución de la etapa de Restauracion de la Base de Datos.

La señal `SIGINT` se enmascara con la función `sig_int1()` que se encarga de interactuar con el usuario para solicitar la interrupción de la carga.

La señal `SIGUSR1` la envía la etapa de Restauración cuando ha terminado su ejecución. Cuando la etapa de Control de Restauración la recibe, termina la etapa Restauración de la Base de Datos. Esta señal se enmascara mediante la función `sig_usr1()`.

Etapa de Restauración. La etapa de Restauración toma como entrada las señales `SIGSTOP`, `SIGCONT` y `SIGTERM` que provienen de la etapa de Control de Restauración.

Esta etapa deshace las modificaciones realizadas en la base de datos por las transacciones New-Order, Payment y Delivery. Para ello, se recorren los campos modificados por cada una de las transacciones y se restaura su valor. Los campos

aleatorios se generan con las mismas semillas que en la carga original.

Para la restauración de la base de datos se utiliza la función `restaura()`, que llama a las funciones `restaura_new_order()`, `restaura_payment()` y `restaura_delivery()`.

Para las transacciones New-Order:

- `d_next_o_id`: se le asigna el valor 2001 en todas las filas de la tabla `district`.
- Se eliminan todas las filas de la tabla New-Order y se vuelven a generar.
- `s_quantity` se le asigna un número aleatorio entre [1,100] mediante aleat_int(), en las filas de la tabla `stock`.
- `s_ytd`, `s_order_cnt` y `s_remote_cnt` se igualan a 0, en las filas de la tabla `stock`.
- Se borran las filas sobrantes de las tablas `orderr` y `order_line`.

Para las transacciones Payment:

- `s_ytd`: se le asigna el valor 300.000, en las filas de la tabla `stock`.
- `d_ytd`: se le asigna el valor 300.000, en las filas de la tabla `district`.
- `c_balance`: se iguala a -10 en las filas de la tabla `customer`.
- `c_ytd_payment`: se iguala a 10 en las filas de la tabla `customer`.
- `c_payment_cnt`: se iguala a 1 en las filas de la tabla `customer`.
- `c_data`: se genera una cadena aleatoria de tamaño comprendido entre [300,500] mediante `cad_alfa_num()`, en las filas de la tabla `customer`.
- Se borran todas las filas de la tabla `history` y se vuelven a generar.

Para las transacciones Delivery:

- El campo `o_carrier_id` se iguala a 0 en todas las filas con el campo `o_id` mayor que 2100.
- El campo `ol_delivery_d` de las filas de la tabla `order_line` cuyo campo `ol_o_id` sea mayor que 2100 se fija a la fecha por defecto para este campo.
- El campo `c_delivery_cnt` de las filas de la tabla `customer` se fija 0.

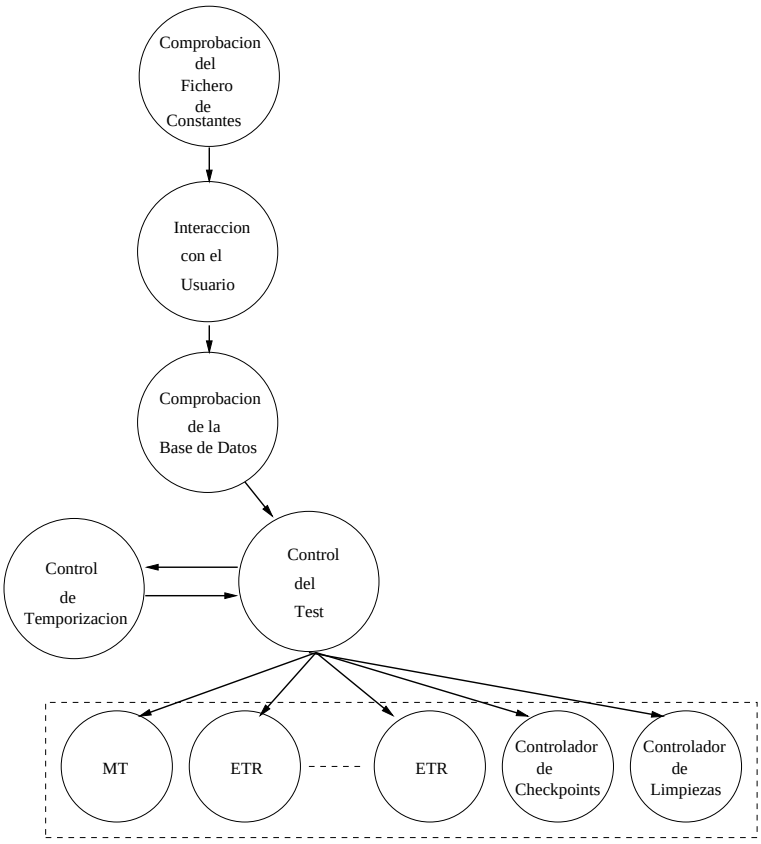


Figura 2.10: Etapa de Ejecución del Test.

Cuando se termina con la restauración de las tablas se realiza una limpieza de la base, de la forma que se explica en el apartado 2.3.1.1.

La función `SIGTERM` se enmascara mediante la función `sig_term_exit()`, que se encarga de finalizar la etapa de restauración.

2.3.2.6 Etapa de Ejecución del Test

Esta etapa se activa cuando el usuario selecciona la opción '3'. Permite realizar un test con los parámetros elegidos por el usuario. En la figura 2.10 se ilustran las operaciones de las que se compone:

Etapa de Comprobación de Fichero de Constantes. Se encarga de comprobar la existencia del fichero de constantes `cons.dat` necesario para que los Emuladores de Terminal Remoto (ETR) generen datos de transacción. Esta comprobación se realiza mediante la función `hay_fich()`.

Si el resultado de la comprobación es erróneo se finaliza la etapa de Ejecución

del Test. En caso contrario se activa la etapa de interacción con el usuario.

Etapa de Interacción con el Usuario. En esta etapa el usuario introduce los datos que configuran el test a realizar. Todas las entradas se comprueban con el fin de evitar realizar un test con parámetros erróneos.

La interacción con el usuario consiste en lo siguiente pasos:

1. Introducción del número de almacenes para realizar el test.
2. Introducción del número de terminales por almacén.
3. Introducción del número del periodo de rampa.
4. Introducción del número del periodo de medida.
5. Confirmación para continuar: esto permite al usuario cancelar la etapa de Ejecución del Test, si considera que los datos introducidos son erróneos. Si el usuario decide continuar se procederá con el paso 6. En caso contrario se finalizará la etapa de Ejecución del Test.
6. Confirmación para realizar limpiezas periódicas en la base de datos: esto permite al usuario decidir si desea que se realicen limpiezas periódicas en la base de datos. En caso de que éste decida no realizarlas se pasará al paso 10. De lo contrario e continua con el paso 7.
7. Introducción del intervalo entre limpiezas.
8. Confirmación para configurar el número máximo de limpiezas en la base de datos. Permitirá al usuario elegir si desea limitar el número de limpiezas a realizar en la base de datos o si de lo contrario desea que se realicen hasta la finalización del test.
9. Introducción del número máximo de limpiezas.
10. Confirmación para continuar: la finalidad de este paso es la misma que la del 5. Si el usuario desea continuar se pasará a la etapa de Comprobación de la Base de Datos. En caso contrario se finalizará la etapa de Ejecución del Test.

Etapa de Comprobación de la Base de Datos. Esta etapa recibe como entrada el número de almacenes con que se desea realizar el test.

Se encarga de hacer las siguientes comprobaciones mediante la función `hay_tablas()`:

- Se comprueba si existen las nueve tablas en la base de datos necesarias para la ejecución del test.
- Se comprueba que existe en la base de datos un número de almacenes igual o mayor que el número seleccionado por el usuario.
- Se comprueba que las tablas poseen la cardinalidad mínima necesaria para la ejecución del test.

Si la comprobación de la base de datos ha resultado exitosa, se activa la etapa de Control del Test. En caso contrario se finalizará la etapa de Ejecución del Test.

Etapa de Control del Test. Se encarga de la activación y de la desactivación de todas las etapas que intervienen directamente en la medida del rendimiento.

Las estradas que recibe esta etapa son:

- De la etapa de Interacción con el Usuario, el número de almacenes con que se desea realizar el test, el número de distritos por almacén, el intervalo de rampa y el intervalo de medida. En el caso de que se haya decidido realizar limpiezas de la base de datos también recibe el intervalo entre estas, y en el caso de que se haya decidido limitar su número, el número máximo.
- Del usuario, la señal de control `SIGINT` cuando este pulsa <ctrl-c>. Esta señal se enmascara mediante la función `signal_int3()` que se encarga de interactuar con el usuario para solicitar la interrupción del test.
- De la etapa de Control de la Temporización, la señal `SIGALARM` que se utiliza para, indicar la expiración de los intervalos de rampa y de medida. Esta señal se enmascara mediante la funcion `sig_alarm()`.

Si durante la etapa de Control del Test, el usuario teclea <ctrl-C>, se saltarán los pasos 4, 5 y 7 que se detallan mas adelante.

A continuación de detallan los pasos de que se compone esta etapa.

1. Se lanza el Monitor de Transacciones (MT) y la población de terminales (ETR). Se lanza tantos terminales por almacén como haya indicado en la entrada. Esto se hace mediante la función `lanzador()`. Esta función se encarga de pasar a cada ETR los parámetros que necesitan para su ejecución que son: llave de la memoria compartida y del semáforo, almacén y distrito al que pertenece, número de almacenes con que se ejecuta el test, y modo de ejecución. Además la función `lanzador()` almacena los identificadores de proceso (pid) del MT y de todos los ETR.

2. Se abre el fichero `medida.log` donde se escriben:
   - Fecha y hora de comienzo del comienzo del test que se obtiene mediante la función `getfechahora()`.
   - Número de almacenes y de terminales por almacén con que se ha realizado el test.
   - Intervalo de rampa y de medida.
   - Sello de hora del comienzo del test.
   - En caso de que se vayan a realizar limpiezas, el intervalo entre ellas.
   - En caso de haberlo configurado, el número máximo de limpiezas.
   - Sello de hora del comienzo del test con precisión de milisegundos.
   - Sello de hora del comienzo del intervalo de medida con precisión de milisegundos.
   - Sello de hora de finalización de intervalo de medida con precisión de milisegundos.
   - Sello de hora de finalización del test con precisión de milisegundos.
3. Se lanza el Controlador de Limpiezas mediante la función `lanza_vacuum()`. En su lanzamiento al Controlador de Limpiezas recibe los parámetros de configuración que son: el intervalo entre limpiezas, y el número máximo.
4. Se activa la etapa de Control de Temporización para que controle el intervalo de rampa, y se espera hasta que esta indique su finalización. Si durante este periodo el usuario teclea <ctrl-c>y decide parar el test, se termina este paso y se saltan los pasos 5 y 7.
5. Se lanza el Controlador de Checkpoints mediante la función `lanza_check()`.
6. Se escribe en `medida.log` el sello de hora de comienzo del intervalo de medida.
7. Se activa la etapa de Control de Temporización para que controle el intervalo de rampa, y se espera hasta que esta indique su finalización. Si durante este periodo el usuario teclea <ctrl-c>y decide parar el test, se termina este paso y se pasa al siguiente.
8. Se escribe en `medida.log` el sello de hora del final del intervalo de medida.
9. Se envía consecutivamente a cada uno de los ETR's la señal `SIGTERM`, para que finalicen su ejecución. En cada envío se espera a que el ETR anterior termine su ejecución.
10. Se envía la señal `SIGTERM` al MT para que finalice su ejecución.
11. Se envía la señal `SIGTERM` al Controlador de Checkpoints para que finalice su ejecución.

12. De haberse lanzado, se envía la señal `SIGTERM` al Controlador de Limpiezas para que finalice su ejecución.

13. Finaliza la Etapa de Ejecución del Test.

Etapa de Control de Temporización. Permite realizar el control de la temporización tanto del intervalo de rampa como del intervalo de medida.

Esta etapa recibe como entradas el intervalo de tiempo que se desea que controle, y una señal de control que le indica el inicio de la temporización.

La salida de esta etapa es la señal de control que indican la expiración del intervalo de tiempo a controlar.

2.3.2.7 Etapa de Comprobación de la Consistencia de la Base de Datos

Esta etapa se activa cuando el usuario selecciona la opción '4'. Se encarga de comprobar las cuatro condiciones de consistencia definidas en las cláusulas 3.3.2.1, 3.3.2.2, 3.3.2.3, y 3.3.2.4 del estándar TPC-C.

Recibe como entrada la señal de control, que indica su inicio.

En esta etapa, primero se cuenta el número de almacenes de que se compone la base de datos, para después pasar a comprobar cada una de las condiciones.

La etapa se implementa con la función `consistencia()`, que a su vez llama a las funciones que se encargan de comprobar las cuatro condiciones. Estas funciones son: `condicion_1()`, `condicion_2()`, `condicion_3()` y `condicion_4()`.

2.3.2.8 Etapa de Eliminación de la Base de Datos

Esta etapa se activa cuando el usuario selecciona la opción '5'. Permite al usuario eliminar la base de datos `tpcc` existente.

Recibe como entrada la señal de control procedente de la etapa menú, que indica su inicio. También recibe la confirmación del usuario para continuar el proceso.

Los pasos que se realizan en esta etapa son:

1. Se pide confirmación al usuario para continuar al proceso. Si este decide no continuar se finaliza la etapa. De lo contrario, se continua con el siguiente paso.

2. Se conecta con `template1` para poder borrar `tpcc`.
3. Se borra `tpcc`.
4. Se desconecta de `template1`.

2.3.2.9 Etapa de Recuento de Resultados del Test

Esta etapa se activa cuando el usuario selecciona la opción '6'. Realiza el análisis de las bitácoras de rendimiento, calculando la tasa tpmC ofrecida por el sistema durante el test, los tiempos de respuesta, los tiempos de pensar y los valores requeridos por las especificaciones del estándar TPC-C que determinan que la carga de trabajo se ha ejecutado correctamente.

La función que realiza esta etapa es `lectura_bitacora()`. Los pasos que se realizan son los siguientes:

1. Se obtienen los parámetros del test leyendo el fichero `medida.log`.
2. Se abren y se preparan los ficheros auxiliares para la realización de los gráficos 1 y 4 definidos en las cláusulas 5.6.1 y 5.6.4 respectivamente.

   Para cada gráfico se crea un fichero auxiliar cuyos registros tienen la estructura siguiente:

   ```
   struct pto {
        float x;
        float y;
   }
   ```

   Se escriben en el fichero tantos registros como intervalos tenga el gráfico correspondiente. En el campo x se representa el comienzo del intervalo.

   En los ficheros auxiliares del gráfico 1, el campo x de los registros tiene un valor que varia desde 0 hasta 4 veces el tiempo de respuesta para el 90 % de las transacciones de cada tipo definido en el estándar TPC-C.

   En el fichero auxiliar del gráfico 4, el campo x varia entre 0 y el tiempo de test expresado en segundos.
3. Se realiza una primera lectura de las bitácoras de rendimiento en la que se obtiene la siguiente información del periodo de medida:
   - Tiempo de respuesta máximo, mínimo y medio de cada transacción.
   - Tiempo de pensar máximo, mínimo y medio de cada transacción.
   - Tiempo de ejecución máximo, mínimo y medio de la transacción Delivery.

- Número total de transacciones.
- Número de transacciones ejecutadas de cada tipo.
- Número de transacciones con un tiempo de respuesta menor al máximo permitido para cada transacción.
- Número de transacciones New-Order canceladas.
- Número de artículos remotos de la transación New-Order.
- Número total de artículos de la transacción New-Order.
- Número de transacciones Payment remotas.
- Número de transacciones Payment y Order-Status cuyo cliente se ha seleccionado por `c_id`.

Además en los ficheros auxiliares de los gráficos 1, para cada transacción se escribe en el campo 'y' el número de transacciones con un tiempo de respuesta perteneciente al intervalo representado por el registro.

En el fichero auxiliar del gráfico 4 se escribe en el campo 'y' de cada registro el número de transacciones New-Order completadas en el intervalo representado por el registro.

4. Se calcula el tiempo de respuesta del 90 % de cada uno de los tipos de transacción. Este paso se realiza gracias a los ficheros auxiliares del gráfico 1. Para cada transacción, se cuenta el campo 'y' de su fichero asociado hasta llegar a alcanzar el 90 % del número total de transacciones. En este punto se anota el valor del campo 'x' del registro siguiente que representa el tiempo de respuesta buscado. Si se alcanza el final de fichero sin haber contado el 90 % del número total de transacciones se rechaza el calculo ya que se considera que sobrepasa excesivamente lo especificado por el estándar TPC-C.

5. Se calcula el número de distritos saltados en las transacciones Delivery. En este paso se lee la bitácora de ejecución de las transacciones Delivery escrita por el Monitor de Transacciones en la que aparecen reflejados los almacenes y distritos involucrados en la transacción.

6. Se analiza la bitácora escrita por el Controlador de Checkpoints para calcular la hora de comienzo y la duración de los cuatro checkpoints más largos. Además se escribe el fichero de salida `check.dat` que contiene los sellos de hora de comienzo y la duración de todos los checkpoints realizados durante el test.

7. Se analiza la bitácora escrita por el Controlador de Limpiezas. Análogamente al paso anterior, se calcula el comienzo y la duración de las cuatro limpiezas más largas y se escribe el fichero `vacuum.dat` que contiene los sellos de hora de comienzo y la duración de todos los checkpoints realizados durante el test.

8. Se Calcula el rendimiento ofrecido por el sistema a a partir del número de transacciones New-Order contadas y del periodo de medida.

9. Se muestra por pantalla un informe con los datos obtenidos en el paso 3, los datos referentes a los checkpoints y las limpiezas obtenidos en los pasos 6 y 7 y el rendimiento ofrecido por el sistema calculado en el paso anterior.

10. Se pide autorización al usuario para escribir los datos que se han mostrado anteriormente en un fichero. Si el resultado es afirmativo, se solicita el nombre del fichero y se escribe. En caso contrario se continua.

11. Se preparan los ficheros auxiliares para la realización de los gráficos 1 y 3 definidos en las cláusulas 5.6.1 y 5.6.3 respectivamente.

    Los ficheros auxiliares del gráfico 1 que se utilizaron para el calculo del tiempo de respuesta del 90 % deben redimensionarse para que los valores del campo 'x' estén en el intervalo de 0 a cuatro veces el valor de tiempo de respuesta del 90 % obtenido, según como se especifica en el estándar TPC-C.

    En el fichero auxiliar del gráfico 3 se escriben 20 registros que representan 20 intervalos de igual longitud desde 0 hasta cuatro veces el tiempo medio de pensar de la transacción New-Order calculado en el paso 3. La estructura de los registros es la misma que la de los ficheros auxiliares de los gráficos 1 y 4.

12. Se realiza una segunda lectura de los ficheros de bitácora. En este paso se obtienen los puntos de los gráficos 1 y 3.

    - Los puntos del gráfico 1 se obtienen de la misma forma que en el paso 3.
    - Los puntos del gráfico 3 se obtienen escribiendo en el campo 'y' de cada registro del fichero auxiliar asociado, el número de New-Orders cuyo tiempo de pensar pertenece al intervalo representado por el registro.

13. Se generan los ficheros de los gráficos 1, 3 y 4. A partir de los ficheros auxiliares se generan los ficheros de texto que contienen los puntos de los gráficos 1, 3 y 4. Se escriben estos ficheros con el formato requerido por el programa visualizador de gráficas Gnuplot.

Los ficheros del gráfico 1 se generan recorriendo el fichero auxiliar asociado y escribiendo el valor de cada uno de los campos del registro en el fichero de texto. Por cada registro escrito, se escribe un salto de línea en el fichero de texto.

El fichero del gráfico 3 se genera análogamente al fichero 1.

El gráfico 4 representa el rendimiento ofrecido por la maquina en función del tiempo. Se obtiene recorriendo el fichero auxiliar del gráfico 4. El valor de cada intervalo se obtiene dividiendo la suma de los campos 'y' de los registros anteriores

por el valor del campo 'x' del registro actual y multiplicando el resultado por 60. En el fichero del gráfico 4, por tanto, se escribe el valor del campo 'x' y el rendimiento obtenido hasta ese punto.

2.3.2.10 Etapa de Comprobación del Estado de la Base de Datos

Esta etapa se activa cuando el usuario selecciona la opción '7'. Imprime por pantalla el número de filas de las nueve tablas de la base de datos, para que el usuario pueda comprobar las modificaciones que se han producido durante un test.

Esta etapa se realiza mediante la función `estado_base()`.

2.3.3. Emulador de Terminal Remoto (ETR)

2.3.3.1 Esquema general

Este módulo tiene como misión realizar la siguientes funciones.

- Simular la introducción de datos por el usuario en la pantalla de E/S del terminal emulado, generando y enviando mensajes de transacción al Monitor de Transacciones.
- Emular la muestra de mensajes de salida en la pantalla de E/S recibiendo y escribiendo los mensajes de respuesta del Monitor de Transacciones.
- Registrar los tiempos de respuesta de transacción.
- Simular los tiempos de espera, teclado y de pensar del usuario emulado.
- Registrar los datos necesarios para el posterior cálculo del rendimiento.

Este programa recibe los siguientes parámetros en su llamada:

- Llave de la memoria y del semáforo de sincronización.
- Almacén al que pertenece el terminal.
- Distrito al que pertenece el terminal.
- Número de almacenes con que se ejecuta el test.

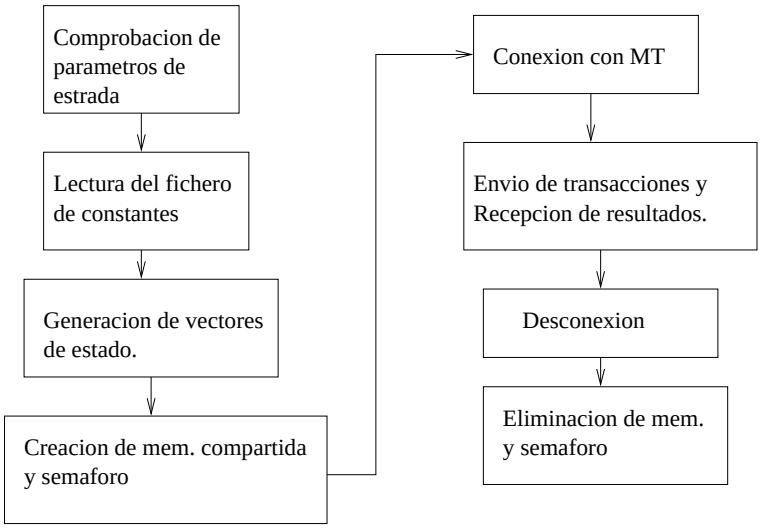


Figura 2.11: Esquema General del ETR

- Modo de ejecución.

  El modo de ejecución determina si las pantallas de terminal se muestran en la salida estándar `stdout` (Modo 0) o por el contrario se envían al dispositivo `/dev/null` (Modo 1). Este ultimo modo es el que se utiliza durante la ejecución del test, con el fin de que no se muestren por pantalla las salidas de los terminales, pero se realice la operación de escritura para simular la actividad de un terminal real.

En la figura 2.11 se muestran las etapas de que se compone este módulo.

A continuación se describe la implementación de cada una de las estas etapas

2.3.3.2 Etapa de Comprobación de los Parámetros de la Llamada

Si en la llamada al programa no se le pasan todos los parámetros, se llama a la función `leyenda()` que avisa por pantalla de tal circunstancia, y se finaliza la ejecución.

2.3.3.3 Etapa Lectura del Fichero de Constantes

En esta etapa se lee el fichero `cons.dat` generado en la creación de la base de datos, que contiene las constantes aleatorias necesarias para la generación los datos de transacción `c_last`, `c_id` y `ol_i_id`. Todos los terminales leen del mismo fichero para que dispongan de las mismas constantes según como define la

cláusula 2.1.6 del estándar TPC-C. Para generar estos campos se usará la función `nurand()` definida en el fichero de cabecera `tpcc.h`, y se le pasará en cada caso la constante correspondiente.

Si el ETR no encuentra este fichero finaliza su ejecución.

2.3.3.4 Etapa de Preparación del Fichero de Bitácora

En el fichero de bitácora es donde se guarda la información de la ejecución de cada transacción. El controlador del bechmark utiliza la información almacenada en este fichero para calcular el rendimiento alcanzado.

En nombre de este fichero se determina a partir del número de almacén y del número de distrito pasados como parámetro. Los nombres de los ficheros de bitácora de los ETR tienen la forma:

`F_W_D.log`

donde `W` y `D` son el número de almacén y de distrito respectivamente.

Estos ficheros se encuentran en el directorio `/tmp` durante el test. Los terminales, al finalizar el test, mueven el fichero al directorio `/usr/share/var/tpcc`. El formato de este fichero se explica en la sección 2.3.1.6.

2.3.3.5 Etapa de Generación de Vectores de Estado

Con el fin de que los clientes generen números aleatorios independientes, se generan unos vectores de estado mediante una semilla obtenida a partir de los números de almacén y de distrito. El primer vector utiliza una semilla formada multiplicando por cien el número de almacén y sumando el número de distrito. Para obtener los siguientes vectores se le va sumando una unidad a la semilla. De esta forma, para cada variable aleatoria, los terminales utilizarán vectores distintos y por tanto se generarán números independientes.

El proceso de creación de vectores de estado se detalla en el apartado 2.3.1.4.

2.3.3.6 Etapa de Creación de la Memoria Compartida y del Semáforo de Sincronismo

El ETR crea la memoria compartida y el semáforo de sincronismo mediante los mecanismos explicados en el apartado 2.3.1.5.

2.3.3.7 Etapa de Conexión

En esta etapa se realiza la conexión entre el ETR y el MT, para permitir la comunicación entre ambos. Para ello el ETR envía al MT un mensaje de conexión.

Los mecanismos de conexión con el Monitor de Transacciones se explican detalladamente en el apartado 2.3.1.5.

2.3.3.8 Etapa de Envío de Transacciones y Recepción de Resultados

Esta etapa, junto con la ejecución de la transacción realizada por el Monitor de Transacciones, son las etapas principales de la realización del test, y son las que determinan los resultados de rendimiento. La etapa de Envío y Recepción es la que genera la carga de trabajo del benchmark.

Los mecanismos de comunicación utilizados en esta etapa se describen en el apartado 2.3.1.5.

El Emulador de Terminal Remoto permanece en esta etapa mientras no se reciba la señal `SIGTERM`. Esta señal proviene del Controlador del Benchmark que la envía cuando ha finalizado el periodo de test. Cuando el ETR recibe esta señal, finaliza la transacción en curso antes de pasar a la etapa de desconexión.

A nivel de código, todo el proceso de envío y recepción se enmarca en un bucle `while` cuya condición de salida está controlada por el flag, que cambia de estado cuando se recibe la señal `SIGTERM`. Para ello esta señal se enmascara con la función, `sigterm()`, que se encarga la modificación del flag, cuando se recibe la señal.

En la figura 2.12 se representan las partes de que se compone la etapa de Envío de Transacciones y Recepción de Resultados. Estas etapas son las siguientes:

Selección de transacción. Esta selección se realiza según un método que permite implementar la mezcla de transacciones especificada en la cláusula 5.2.3 del estándar TPC-C. El método utilizado se basa en el método de la baraja de la cláusula 5.2.4.2 del estándar TPC-C.

La selección se realiza generando un vector de 23 enteros que identifican a cada transacción, al que se denomina *baraja*. Los identificadores de transacción son constantes declaradas en el fichero de cabecera `tpcc.h` del benchmark. En el vector se distribuyen:

- 10 identificadores de transaccion New-Order.

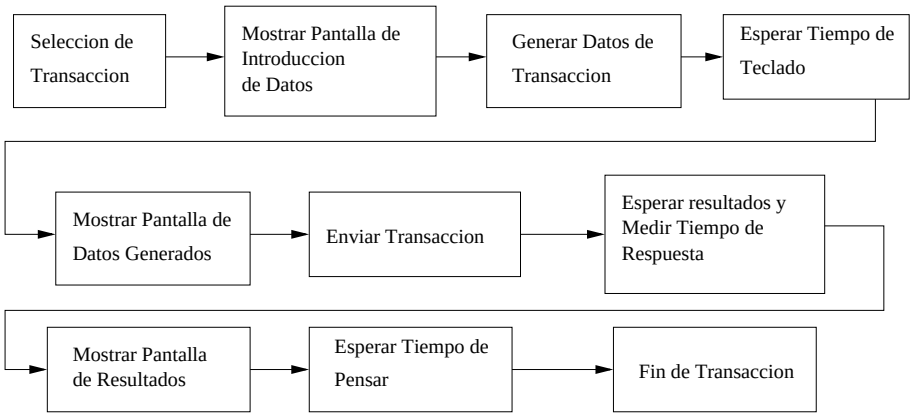


Figura 2.12: Etapa de Envío de Mensajes y Recepción de Resultados

- 10 identificadores de transaccion Payment.
- 1 identificador de transaccion Stock-Level.
- 1 identificador de transaccion Delivery.
- 1 identificador de transaccion Order-Status.

La mezcla anterior proporciona:

- el 43,47 % de transacciones New-Order.
- el 43,47 % de transacciones Payment.
- el 4,35 % de transacciones Stock-Level.
- el 4,35 % de transacciones Delivery.
- el 4,35 % de transacciones Order-Status.

de esta forma se cumplen los porcentajes especificados en la cláusula 5.2.3 del TPC-C.

Para implementar la selección aleatoria de transacciones, a las posiciones de la baraja se accede a través de un vector aleatorio de apuntadores, que se actualiza cada vez que se han accedido a todas las posiciones de la baraja. La actualización del vector de apuntadores realiza mediante la función `posicion_cartas()` definida en el fichero de cabecera `tpcc.h`. La figura 2.13 ilustra el mecanismo de selección de transacción.

El número de identificación del tipo de transacción seleccionada se escribe en el fichero de bitácora, además del sello de inicio de la transacción. Tras lo anterior la ejecución de la transacción se discrimina mediante un `switch`.

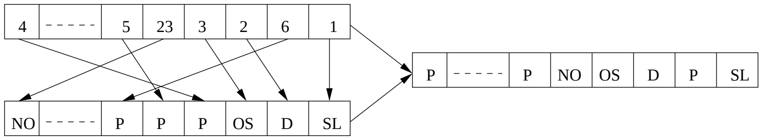


Figura 2.13: Mecanismo de Selección de Transacciones

Muestra de la pantalla de introducción de datos. En este punto se muestra la pantalla de introducción de datos para cada tipo de transacción según se especifica en los perfiles de transacción definidos en la cláusula 2 del estándar TPC-C. Estas pantallas son las siguientes:

Transación New-Order. Se muestra la siguiente pantalla por medio de la función `pant_new_order_pet()`:

```
                                    New Order
Warehouse:    1   District: ##                     Date: YYYY-DD-MM hh:mm:ss
Customer:  ####   Name: ----------------   Credit: --   %Disc: --,--
Order Number: --------  Number of Lines: --       W_tax: --,--   D_tax: --,--

 Supp_W  Item_Id  Item Name                 Qty  Stock  B/G  Price    Amount
  ####   ######   ------------------------  ##    ---    -   ---,--E  ----,--E
  ####   ######   ------------------------  ##    ---    -   ---,--E  ----,--E
  ####   ######   ------------------------  ##    ---    -   ---,--E  ----,--E
  ####   ######   ------------------------  ##    ---    -   ---,--E  ----,--E
  ####   ######   ------------------------  ##    ---    -   ---,--E  ----,--E
  ####   ######   ------------------------  ##    ---    -   ---,--E  ----,--E
  ####   ######   ------------------------  ##    ---    -   ---,--E  ----,--E
  ####   ######   ------------------------  ##    ---    -   ---,--E  ----,--E
  ####   ######   ------------------------  ##    ---    -   ---,--E  ----,--E
  ####   ######   ------------------------  ##    ---    -   ---,--E  ----,--E
  ####   ######   ------------------------  ##    ---    -   ---,--E  ----,--E
  ####   ######   ------------------------  ##    ---    -   ---,--E  ----,--E
  ####   ######   ------------------------  ##    ---    -   ---,--E  ----,--E
  ####   ######   ------------------------  ##    ---    -   ---,--E  ----,--E
  ####   ######   ------------------------  ##    ---    -   ---,--E  ----,--E
Execution Status: ------------------------                   Total:  -----,--E
Select next transaction type: N (New-Order), P (Payment), S (Order-Status),
                              D (Delivery), L (Stock-Level). SELECTION:  N
```

Transación Payment. Se muestra la siguiente pantalla por medio de la función `pant_payment_pet()`:

```
                                    Payment
Date: YYYY-DD-MM hh:mm:ss

Warehouse:     1                          District: ##
--------------------                     --------------------
--------------------                     --------------------
-------------------- -- ----------       -------------------- -- ----------

Customer: ####  Cust-Warehouse: ####  Cust-District: ##
Name:   ---------------- -- ----------------     Since:  YYYY-DD-MM
        --------------------                     Credit: --
        --------------------                     %Disc:  --,--
        -------------------- -- ----------       Phone:  -------------------

Amount Paid:           ####,##E      New-Cust-Balance: -----------,--E
Credit Limit:   ----------,--E

Cust-Data: --------------------------------------------------
           --------------------------------------------------
           --------------------------------------------------
           --------------------------------------------------
```

Transación Order-Status. Se muestra la siguiente pantalla por medio de la función `pant_ostatus_pet()`:

```
                                Order-Status
Warehouse:     1   District: ##
Customer: ####   Name: ---------------- -- ################
Cust-Balance: ------,--E

Order Number: --------   Entry-Date: YYYY-MM-DD hh:mm:ss   Carrier-Number: --
Supp_W       Item_Id    Qty     Amount      Delivery-Date
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
Select next transaction type: N (New Order), P (Payment), S (Order Status),
                              D (Delivery), L (Stock Level). SELECTION:  S
```

Transación Delivery. Se muestra la siguiente pantalla por medio de la función `pant_delivery_pet()`:

```
                                  Delivery
Warehouse:    1

Carrier Number: ##

Execution Status: -------------------------









Select next transaction type: N (New Order), P (Payment), S (Order Status),
                              D (Delivery), L (Stock Level). SELECTION:  D
```

Transación Stock-Level. Se muestra la siguiente pantalla por medio de la función `pant_stock_level_pet()`:

```
                                 Stock-Level
Warehouse:    1   District:  1

Stock Level Threshold: ##

low stock: ---









Select next transaction type: N (New-Order), P (Payment), S (Order-Status),
                              D (Delivery), L (Stock-Level). SELECTION:  S
```

Generación de Datos de Transacción. En esta etapa se generan los datos de transacción especificados en los perfiles de transacción del estándar TPC-C. Cada dato generado se valida para simular las operaciones de comprobación de los campos de entrada de un terminal real, según se especifica en la cláusula

2.2.2.4 del TPC-C. A continuación se detalla cómo se generan los datos para cada transacción.

Transacción New-Order:

En cada New_Order se generan los siguientes datos, que son introducidos en la estructura de mensaje de transacción:

Campo	Mecanismo de generación
w_id	Se introduce el almacén al que pertenece el terminal.
d_id	Se selecciona entre 1 y 10 mediante la función `aleat_int()`.
c_id	Se selecciona utilizando la función `nurand()`.
i_id	Se selecciona utilizando la función `nurand()`

El número de artículos de que se compone la orden se determina seleccionándolo mediante la función `aleat_int()` definida en la librería `tpcc.h`. En el 1 % de las transacciones, la ultima línea de la orden debe contener un identificador de artículo no válido. Para ello se genera un número entre 1 y 100 con la función `aleat_int()` que determinara este caso. Si el número resultante es 1 la última línea de la orden contendrá un número de artículo no válido.

El 1 % de los artículos deben pertenecer a un almacén distinto al que pertenece el terminal (artículos remotos). Para implementar esta característica se genera un numero aleatorio entre 1 y 100 mediante la función `aleat_int()`. Si el resultado es 1 se da esta circunstancia. En el caso de que el test se realice con un sólo almacén, no se considera esta posibilidad.

Para cada articulo de la orden, se generan los siguientes campos:

Campo	Mecanismo de generación
ol_i_id	Se selecciona utilizando la función `nurand()`, según la cláusula 2.4.1.2 del TPC-C. Si es el último artículo y este ha de ser inválido se introduce el valor de la constante ART_UNUSED definida en tpcc.h
c_id	Se selecciona utilizando la función `nurand()`, según la cláusula 2.4.1.5 del TPC-C.
ol_supply_w_id	Se introduce el identificador del almacén local el 99 % de los casos En el 1 % restante se selecciona al azar el identificador de uno de los almacenes restantes mediante la función `aleat_int()`.
ol_quantity	Se selecciona entre 1 y 10 mediante la función `aleat_int`.

Transacción Payment:

En cada Payment se generan los siguientes datos, que son introducidos en la correspondiente estructura de mensaje de transacción:

Campo	Mecanismo de generación
w_id	- Se introduce el almacén al que pertenece el terminal.
d_id	- Se selecciona entre 1 y 10 mediante la función `aleat_int()`.

En el 40 % de las transacciones el cliente debe seleccionarse mediante el campo `c_id`, y en el 60 %restante mediante el campo `c_last`. Para identificar cada caso en cada transacción se genera una número aleatorio entre 0 y 100. Si el resultado es menor o igual que 40 se estará en el primer caso, de lo contrario en el segundo.

Además en el 15 % de las transacciones el cliente debe seleccionarse de un almacén y un distrito al que pertenece el terminal. Cuando se da este caso se dice que la transacción Payment es *remota*, de lo contrario se dice que es *local*. Esto se implementa generando un número aleatorio entre 1 y 100. Si el resultado es mayor que 85 se da esta circunstancia. Por lo tanto:

Campo	Mecanismo de generación
c_id	Si el cliente se selecciona mediante este campo se introduce un número generado mediante la función `nurand()`. De lo contrario se introduce '0'.
c_last	Si el cliente se selecciona mediante este campo se introduce una cadena aleatoria generada mediante la función `crea_clast()` a partir de un número generado mediante la función `nurand()`, según se especifica en la cláusula 2.5.1.2 de las especificaciones del TPC-C. De lo contrario se introduce '\0'.
c_w_id	Si el cliente es remoto se le asigna un identificador de almacén seleccionado al azar de entre los almacenes configurados en el test. De lo contrario se le asigna el número de almacén al que pertenece el terminal. Si en el test sólo hay un almacén no se considera esta posibilidad.
c_d_id	Si el cliente es remoto se le asigna un identificador de distrito seleccionado al azar mediante la función `aleat_int()`. De lo contrario se le asigna el distrito al que pertenece el terminal.
h_amount	Se le asigna un número seleccionado al azar entre 1 y 5000 mediante la función `aleat_dbl()`.

Transacción Order-Status:

En cada transacción Order-Status se generan los siguientes datos, que son introducidos en la estructura de mensaje de transacción:

Campo	Mecanismo de generación
w_id	Se introduce el almacén al que pertenece el terminal
d_id	Se le asigna un número aleatorio entre 1 y 10 obtenido mediante la función aleat_int()

A igual que en la transacción Payment, en el 40 % de las transacciones el cliente

debe seleccionarse mediante el campo `c_id`, y en el 60 % restante meciante el campo `c_last`. Esto se identifica de la misma forma que en el caso anterior. Por lo tanto:

Campo	Mecanismo de generación
c_id	Si el cliente se selecciona mediante este campo se introduce un número generado mediante la función `nurand()`. De lo contrario se introduce '0'.
c_last	Si el cliente se selecciona mediante este campo se introduce una cadena aleatoria generada mediante la función `crea_clast()` a partir de un número generado mediante la función `nurand()`, según se especifica en la cláusula 2.5.1.2 de las especificaciones del TPC-C. De locontrario se introduce '\0'.

Transacción Delivery:

En cada transacción Delivery se generan el siguiente dato que son introducidos en la estructura de mensaje de transacción:

Campo	Mecanismo de generación
o_carrier_id	se introduce un número seleccionado al azar entre 1 y 10 obtenido mediante la función `aleat_int()`.

Transacción Stock-Level:

Para la transacción Stock-Level se generan los siguientes datos que son introducidos en la estructura de datos de transacción:

Campo	Mecanismo de generación
w_id	Se introduce el almacén al que pertenece el terminal.
d_id	Se introduce el distrito al que pertenece el terminal.

Esperar tiempo de teclado. En esta etapa se esperan los tiempos de teclado definidos en la cláusula 5.2.5.7 de estándar TPC-C para cada transacción. Se implementa utilizando la función `sleep()` [17] de C. Los tiempos de espera de cada transacción están definidos como constantes en el fichero de cabecera tpcc.h.

Muestra de pantalla de datos introducidos. En este paso se muestran las pantallas con los datos generados para cada transacción, según como se define en los perfiles de transacción de la cláusula 2 del TPC-C.

Transación New-Order. Se muestra la siguiente pantalla por medio de la función `pant_new_order_muest()`:

```
                                New Order
Warehouse:     1   District:  8                    Date: YYYY-DD-MM hh:mm:ss
Customer:  1314   Name: ----------------   Credit: --   %Disc: --,--
Order Number: --------  Number of Lines: --        W_tax: --,--   D_tax: --,--

 Supp_W  Item_Id  Item Name                 Qty  Stock  B/G  Price    Amount
     1    15429   ------------------------   9    ---    -   ---,--E  ----,--E
     1    31828   ------------------------   5    ---    -   ---,--E  ----,--E
     1    24874   ------------------------   1    ---    -   ---,--E  ----,--E
     1    34425   ------------------------   9    ---    -   ---,--E  ----,--E
     1    15998   ------------------------   9    ---    -   ---,--E  ----,--E
     1    50549   ------------------------   2    ---    -   ---,--E  ----,--E
     1    10344   ------------------------   6    ---    -   ---,--E  ----,--E
     1    67010   ------------------------   7    ---    -   ---,--E  ----,--E
     1    37011   ------------------------   7    ---    -   ---,--E  ----,--E
     1    89178   ------------------------   7    ---    -   ---,--E  ----,--E
     1    71800   ------------------------   6    ---    -   ---,--E  ----,--E
     1    48244   ------------------------   5    ---    -   ---,--E  ----,--E


Execution Status: ------------------------                 Total:  -----,--E
Select next transaction type: N (New-Order), P (Payment), S (Order-Status),
                              D (Delivery), L (Stock-Level). SELECTION:  N
```

Transación Payment. Se muestra la siguiente pantalla por medio de la función `pant_payment_muest()`:

```
                                  Payment
Date: YYYY-DD-MM hh:mm:ss

Warehouse:     1                         District:  2
--------------------                     --------------------
--------------------                     --------------------
-------------------- -- ----------       -------------------- -- ----------

Customer: ####  Cust-Warehouse:     1  Cust-District:  2
Name:   ---------------- --  ABLEBARPRI          Since:  YYYY-DD-MM
        --------------------                     Credit: --
        --------------------                     %Disc:  --,--
        -------------------- -- ----------       Phone:  --------------------

Amount Paid:          1395,46E      New-Cust-Balance: -----------,--E
Credit Limit:   ----------,--E

Cust-Data: --------------------------------------------------
           --------------------------------------------------
           --------------------------------------------------
           --------------------------------------------------
```

Transación Order-Status. Se muestra la siguiente pantalla por medio de la función `pant_ostatus_muest()`:

```
                               Order-Status
Warehouse:     1   District:  4
Customer: 1079   Name: ---------------- -- ################
Cust-Balance: ------,--E

Order Number: --------   Entry-Date: YYYY-MM-DD hh:mm:ss   Carrier-Number: --
Supp_W       Item_Id    Qty     Amount      Delivery-Date
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
  ----       ------     --     -----,--E      YYYY-MM-DD
Select next transaction type: N (New Order), P (Payment), S (Order Status),
                              D (Delivery), L (Stock Level). SELECTION:  S
```

Transación Delivery. Se muestra la siguiente pantalla por medio de la función `pant_delivery_muest()`:

```
                                   Delivery
Warehouse:    1
Carrier Number:  5
Execution Status: -------------------------









Select next transaction type: N (New-Order), P (Payment), S (Order-Status),
                              D (Felivery), L (Stock-Level). SELECTION: D
```

Transación Stock-Level. Se muestra la siguiente pantalla por medio de la función `pant_stock_level_muest()`:

```
                                   Stock-Level
Warehouse:    1   District:  1
Stock Level Threshold: 14
low stock: ---









Select next transaction type: N (New-Order), P (Payment), S (Order-Status),
                              D (Delivery), L (Stock-Level). SELECTION:  S
```

Envío de transacción. Tras haber generado los datos en un paso anterior, se escribe en el fichero de bitácora el sello de hora de envío de transacción. Tras esto, los datos generados se envían al Monitor de Transacciones a través de la cola de mensajes, como se explica en el apartado 2.3.1.5.

Esperar resultados y medir tiempo de respuesta. En este punto el ETR espera a que el Monitor de Transacciones responda con los resultados de la ejecución de la transacción. Cuando el MT abra el semáforo de sincronismo el ETR podrá acceder a la memoria compartida.

Cuando el ETR accede a los resultados, mide el tiempo de respuesta transcurrido desde el envío de la transacción hasta la recepción de la respuesta. El tiempo de respuesta se calcula tomando un sello de hora tras haber recibido el mensaje de respuesta y restándolo con el tomado antes de enviar la transacción. Esta resta se efectúa mediante la función `resta_tiempos()` declarada en el fichero de cabecera `tpcc.h`.

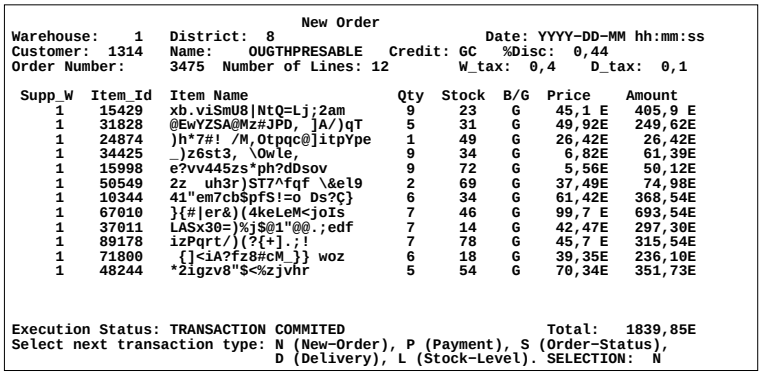
En el caso de la transacción Delivery no se espera a que el Monitor de Transacciones responda ya que su perfil de transacción especifica que únicamente ha

de ser encolada. En este caso el tiempo que se mide es el tiempo de encolado.

El tiempo de respuesta se registra en el fichero de bitácora.

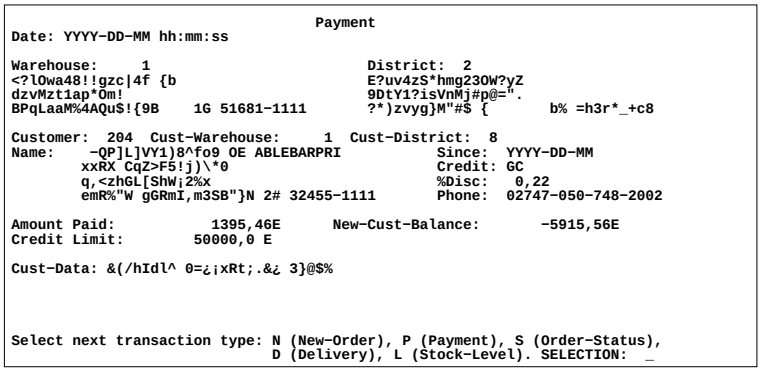
Muestra de la pantalla de resultados de transacción. En este paso se muestran las pantallas con los datos resultantes de la ejecución de la transacción y el campo de selección de menú, según como se define en los perfiles de transacción de la cláusula 2 del TPC-C.

Transación New-Order. Se muestra la siguiente pantalla por medio de la función `pant_new_order_menu()`:



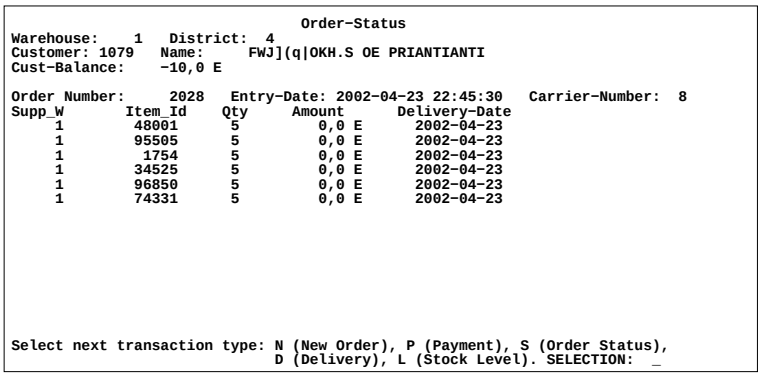
```
                                New Order
Warehouse:    1   District:  8                    Date: YYYY-DD-MM hh:mm:ss
Customer:  1314   Name:    OUGTHPRESABLE   Credit: GC   %Disc:  0,44
Order Number:     3475  Number of Lines: 12        W_tax:  0,4    D_tax:  0,1

 Supp_W  Item_Id  Item Name                 Qty  Stock  B/G  Price    Amount
     1    15429   xb.viSmU8|NtQ=Lj¡2am      9     23    G    45,1 E   405,9 E
     1    31828   @EwYZSA@Mz#JPD, jA/)qT     5     31    G    49,92E   249,62E
     1    24874   )h*7#! /M,Otpqc@]itpYpe    1     49    G    26,42E    26,42E
     1    34425   _)z6st3, \Owle,            9     34    G     6,82E    61,39E
     1    15998   e?vv445zs*ph?dDsov         9     72    G     5,56E    50,12E
     1    50549   2z  uh3r)ST7^fqf \&el9     2     69    G    37,49E    74,98E
     1    10344   41"em7cb$pfS!=o Ds?Ç}      6     34    G    61,42E   368,54E
     1    67010   }{#|er&)(4keLeM<joIs       7     46    G    99,7 E   693,54E
     1    37011   LASx30=)%j$@1"@@.;edf      7     14    G    42,47E   297,30E
     1    89178   izPqrt/)(?{+].;!           7     78    G    45,7 E   315,54E
     1    71800    {]<iA?fz8#cM_}} woz       6     18    G    39,35E   236,10E
     1    48244   *2igzv8"$<%zjvhr           5     54    G    70,34E   351,73E



Execution Status: TRANSACTION COMMITED                      Total:   1839,85E
Select next transaction type: N (New-Order), P (Payment), S (Order-Status),
                              D (Delivery), L (Stock-Level). SELECTION:  N
```

Transación Payment. Se muestra la siguiente pantalla por medio de la función `pant_payment_menu()`:

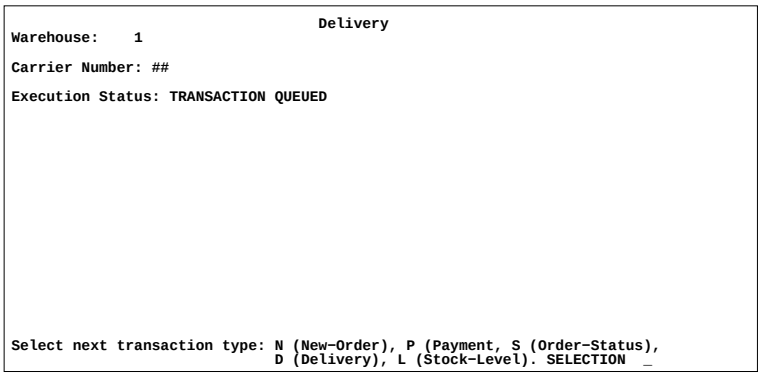


```
                                  Payment
Date: YYYY-DD-MM hh:mm:ss

Warehouse:     1                        District:  2
<?lOwa48!!gzc|4f {b                     E?uv4zS*hmg23OW?yZ
dzvMzt1ap*Om!                           9DtY1?isVnMj#p@=".
BPqLaaM%4AQuS!{9B    1G 51681-1111      ?*)zvyg}M"#$ {       b% =h3r*_+c8

Customer:  204  Cust-Warehouse:     1  Cust-District:  8
Name:    -QP]L]VY1)8^fo9 OE ABLEBARPRI          Since:  YYYY-DD-MM
         xxRX CqZ>F5!j)\*0                     Credit: GC
         q,<zhGL[ShW¡2%x                       %Disc:   0,22
         emR%"W gGRmI,m3SB"}N 2# 32455-1111    Phone:  02747-050-748-2002

Amount Paid:           1395,46E      New-Cust-Balance:       -5915,56E
Credit Limit:        50000,0 E

Cust-Data: &(/hIdl^ 0=¿¡xRt;.&¿ 3}@$%



Select next transaction type: N (New-Order), P (Payment), S (Order-Status),
                              D (Delivery), L (Stock-Level). SELECTION:  _
```

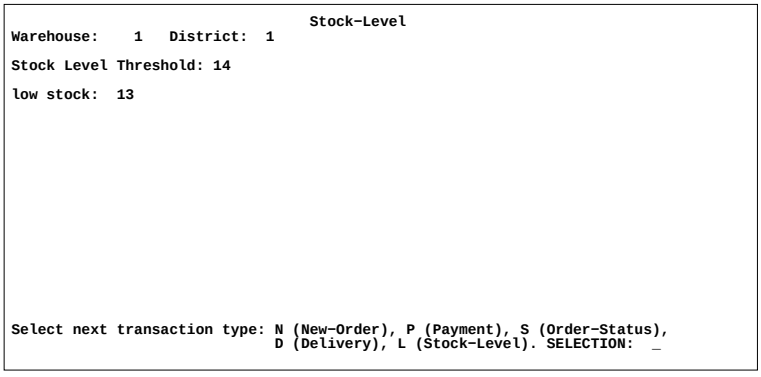
Transación Order-Status. Se muestra la siguiente pantalla por medio de la función `pant_ostatus_menu()`:



Transación Delivery. Se muestra la siguiente pantalla por medio de la función `pant_delivery_menu()`:



Transación Stock-Level. Se muestra la siguiente pantalla por medio de la función `pant_stock_level_menu()`:



Esperar tiempo de pensar. El tiempo de pensar se genera mediante la función `aleat_tpensar()` que implementa el método definido en la cláusula 5.2.5.4 del estándar TPC-C. En la cláusula 5.2.5.7 se definen la media mínima de la distribución:

- 12 s para la transacción New-Order.
- 12 s para la transacción Payment.
- 10 s para la transacción Order-Status.
- 5 s para la transacción Delivery.
- 5 s para la transacción Stock-Level.

La media para cada tipo de transacción se define como constante en el fichero de cabecera `tpcc.h`.

El tiempo de pensar generado se registra en el fichero de bitácora.

Finalización de la transacción. En este paso se escriben en el fichero de bitácora tres cifras cuyo significado varía según la transacción realizada (ver formato de fichero de bitácora de ETR en la sección 2.3.1.6).

- New-Order.
  - La primera cifra indica si la transacción ha sido cancelada como resultado de enviar un número de artículo no válido (ver definición de artículo no válido en el perfil de transacción de la cláusula 2.4 del TPC-C), en cuyo caso se escribe un '-1'. Si la transacción se ha confirmado se escribe un '0'.
  - La segunda cifra indica el número de artículos de que se componía la orden.
  - La tercera cifra indica el número de artículos remotos (ver definición de artículo no válido en el perfil de transacción de la cláusula 2.4) que contenía la orden.
- Payment.
  - La primera cifra no contiene información y se escribe siempre un '0'.
  - La segunda cifra indica si la transacción Payment es remota (ver definición de transacción remota en el perfil de transacción de la cláusula 2.5 del TPC-C), en cuyo caso se escribe un '1'. En caso contrario un '0'.
  - La tercera cifra indica si el cliente se ha seleccionado por `c_id` (ver perfil de transacción de la cláusula 2.5), en cuyo caso se escribe un '1'. En caso contrario un '0'.
- Order-Status.

  - La primera cifra no contiene información y se escribe siempre un '0'.
  - La segunda cifra indica si el cliente se ha seleccionado por `c_id` (ver perfil de transacción de la cláusula 2.6 del TPC-C), en cuyo caso se escribe un '1'. En caso contrario un '0'.
  - La tercera cifra no contiene información y se escribe siempre un '0'.

- Delivery y Stock-Level.

  - Las cifras no contienen información por lo que se escribe siempre un '0'.

En este paso también se escribe el sello de hora de finalización de la transacción.

Si durante la transacción en curso, el ETR no ha recibido la señal `SIGTERM` enviada por el Controlador del Benchmark, el ETR, regresa a la etapa de Selección de Transacción. De lo contrario pasa a la etapa de Desconexión. La señal `SIGTERM` avisa del fin del periodo de medida.

2.3.3.9 Etapa de Desconexión

Tras recibir la señal `SIGTERM`, el terminal espera a la finalización de la transacción en curso y se desconecta del Monitor de Transacciones enviándole un mensaje de desconexión tal y como se explica en el apartado 2.3.1.5.

2.3.3.10 Etapa de eliminación de la memoria compartida y del semáforo de sincronismo

El ETR elimina la memoria compartida y el semáforo de sincronismo mediante los mecanismos descritos en el apartado 2.3.1.5.

2.3.4. Monitor de Transacciones (MT)

2.3.4.1 Esquema general

Este módulo tiene como misión realizar las siguientes funciones:

- Se encarga de ejecutar las solicitudes de transacción que le envia la población Emuladores de Terminal Remoto.

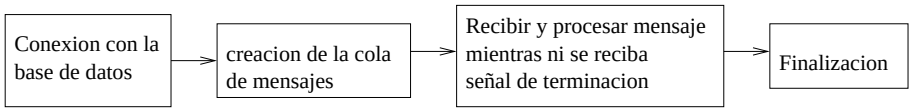


Figura 2.14: Esquema del Monitor de Transacciones

- Registra los posibles errores resultantes de la ejecución de transacciones en la bitácora de error `tm_err.log`.

- Registra los resultados de la ejecución aplazada de las transacciones Delivery, según se especifica en la cláusula 2.7 del estándar TPC-C, en la bitácora `tm_delivery_res.log`.

- Registra los tiempos de respuesta de la transacción Delivery, para el posterior cálculo del rendimiento, en la bitácora `tm_delivery_tr.log`.

En la figura 2.14 se representan las etapas de funcionamiento del Monitor de Transacciones.

Los mecanismos de recepción de mensajes y envío de respuestas por parte de MT se explican detalladamente en el apartado 2.3.1.5.

2.3.4.2 Etapa de conexión con la base de datos

Se realiza una conexión con postgtreSQL, a través de la cual se ejecutan todas las transacciones que recibe el Monitor de Transacciones.

En esta etapa también se activa el modo *autocommit* en la conexión, con el fin de poder confirmar ('commit') o cancelar ('rollback') las transacciones en curso.

Las consideraciones acerca de la conexión con postgreSQL y la activación del modo autocommit se explican en el apartado 2.3.1.1.

2.3.4.3 Etapa de Creación de la Cola de Mensajes

En esta etapa el Monitor de Transacciones crea la cola de mensajes por donde se reciben las transacciones enviadas por la población de Emuladores de Terminal Remoto, así como los mensajes de conexión y desconexión.

Los mecanismos para la creación se explican en el apartado 2.3.1.5.

2.3.4.4 Etapa de Recepción y Tratamiento de Mensajes

En este momento, el programa espera a la recepción de los mensajes procedentes de la población de ETRs. Cada mensaje se discrimina mediante un `switch` que identifica el tipo de mensaje recibido. Tras recibirlo y tratarlo, se vuelve a esperar a la recepción de un nuevo mensaje. Esto se implementa por medio de un bucle `while` cuya condición de permanencia está determinada por un flag. Éste se modifica cuando se recibe la señal `SIGTERM`, enmascarada mediante la función `sigterm()`. Durante el test será el Controlador del Benchmark quien envíe esta señal cuando éste finalice.

Los mecanismos de sincronización entre ETR y MT en esta etapa se explican en el apartado 2.3.1.5.

Los tipos de mensajes que recibe el Monitor de Transacciones:

- Mensaje de terminal: A través de este mensaje el ETR solicita la conexión y la desconexión con el Monitor de Transacciones.
- Mensaje de transacción New-Order: Representa un mensaje de transacción New-Order.
- Mensaje de transacción Payment: Representa un mensaje de transacción Payment.
- Mensaje de transacción Order-Status: Representa un mensaje de transacción Order-Status.
- Mensaje de transacción Delivery: Representa un mensaje de transacción Delivery.
- Mensaje de transacción Stock-Level: Representa un mensaje de transacción Stock-Level.

Las operaciones que se realizan para cada tipo de mensaje recibido se describen a continuación:

Mensaje de terminal

Cuando se recibe un mensaje de este tipo, se discrimina entre mensajes de conexión y mensajes de desconexión. Permitirán que un nuevo ETR se conecte con el MT, o que un ETR ya conectado se desconecte.

Los mecanismos para la conexión y desconexión de los ETR se detallan en el apartado 2.3.1.5.

Mensaje de transacción New-Order

Al recibir este mensaje, el Monitor de Transacciones ejecuta una transacción New-Order con los datos recibidos en la estructura de mensaje de transacción, siguiendo el perfil de la cláusula 2.4 del estándar TPC-C. La función que se encarga de la ejecución de la transacción es `trans_new_order()`. A esta función se le pasa como parámetros la estructura de datos de transacción recibida y el puntero a la memoria compartida. A medida que se va ejecutando la transacción se escriben los resultados en la memoria compartida. La estructura de datos de mensaje de transacción New-Order y la estructura de mensaje de respuesta se describen en el apartado 2.3.1.5. Por cada mensaje recibido se imprime un mensaje en pantalla.

A continuación se detalla la implementación del perfil de la transacción New-Order de la cláusula 2.4 del TPC-C.

1. Se obtiene la fecha del sistema mediante la función `getfechahora()`, se almacena en la variable compartida SQL `o_entry_d` y se escribe en la memoria compartida.

2. Se extraen los datos de la estructura de mensaje de transacción y se almacenan en variables compartidas SQL.

3. En las tablas `warehouse` y `customer`, se seleccionan los atributos `w_tax`, `c_discount`, `c_credit` de la fila cuyos campos `w_id`, `c_w_id` y `c_d_id` coinciden con los valores recibidos en el mensaje de transacción. Los valores obtenidos se almacenan en variables compartidas SQL y se escriben en la memoria compartida.

4. En la tabla `district` se seleccionan los atributos `d_next_o_id` y `d_tax` de la fila cuyos campos `d_w_id` y `d_id` coinciden con los valores recibidos en el mensaje de transacción. Los valores obtenidos se almacenan en variables compartidas SQL y se escriben en la memoria compartida.

5. En la misma fila de la tabla `district` seleccionada en el paso anterior, se actualiza el atributo `d_next_o_id` incrementándolo en una unidad.

6. Se inserta una nueva fila en la tabla `orderr` con los valores `o_id` igual al valor obtenido del atributo `d_next_o_id` en el paso 4, `o_d_id`, `o_w_id` y `o_c_id` iguales a los valores recibidos en la estructura de mensaje de transacción, `o_carrier_id` igual al valor 0, `o_entry_d` igual a la fecha obtenida en el paso 1, `o_all_local` igual al valor 1 y `o_carrier_id` igual al valor 0.

7. Se inserta una nueva fila en la tabla `new_order` con los atributos `no_o_id` igual al valor `d_next_o_id` obtenido en el paso 4, `no_w_id`, `no_d_id` iguales a los recibidos en la estructura de mensaje de transacción.

8. Para cada uno de los artículos contenidos en el mensaje de transacción, se realizan las siguientes operaciones:

a) Se extraen los datos que identifican a cada una de las líneas de orden de la estructura de mensaje y se almacenan en variables compartidas SQL.

b) De la tabla `item` se seleccionan los atributos `i_price`, `i_name` e `i_data` de la fila cuyo campo `i_id` coincide con el valor `ol_i_id` extraído de la estructura del mensaje. Los valores obtenidos se almacenan en variables compartidas SQL y se escriben en la memoria compartida. Si no se encuentra al artículo buscado se cancela la transacción mediante `EXEC SQL ROLLBACK`, según se especifica en perfil de la transacción. Para identificar este caso se utiliza el error devuelto por postgreSQL en la estructura `sqlca`.

c) De la tabla `stock` se seleccionan `s_quantity`, `s_data` y `s_dist_01`... `s_dist_10` de las filas cuyos campos, `s_i_id`, `s_w_id` coinciden con los valores `ol_i_id` y `ol_supply_w_id` obtenidos del mensaje de transacción. El valor del atributo `s_data` obtenido se almacena en una variable compartida SQL y se escriben en la memoria compartida.

d) Se escribe en la memoria compartida el atributo `s_dist_XX` del distrito indicado por el campo `d_id` del mensaje de transacción.

e) Se actualiza el campo `s_quantity` de la tabla `stock` según se define en el perfil de transacción.

f) Si el artículo actual no es local se suma una unidad el atributo `s_remote_cnt` de la fila de la tabla `stock` cuyos campos `s_i_id` y `s_w_id` sean iguales a los campo `s_i_id` y `s_supply_w_id` de la estructura de mensaje de transacción para el artículo en curso. Además se fuerza a cero el flag que determina si todos los artículos son locales

g) Si el campo `i_data` obtenido en el paso 8.b contiene la cadena `ORIGINAL`, se escribe el caracter 'B' en el campo `b_g` de la memoria compartida para el artículo en curso.

h) Se calcula el valor de `ol_amount` multiplicando el valor de `ol_cuantity` por el de `i_price`.

i) Se calcula la suma parcial de los valores de `ol_amount`.

j) Se inserta una nueva línea en la tabla `order_line` con los valores `ol_i_id` igual al valor `d_next_o_id` obtenido en el paso 4, `ol_w_id`, `ol_d_id`, `ol_i_id` `ol_supply_w_id` y `ol_quantity` iguales a los valores de los campos correspondientes en el mensaje de transacción, `ol_number` igual al número de línea del artículo en curso, ol_amount obtenido en el paso anterior y `ol_dist_info` obtenido en el paso 8.d

9. Si algún artículo es remoto se escribe un cero en el campo `o_all_local` de la fila de la tabla `orderr` cuyos campos `o_id` y `o_d_id` son iguales a los valores `d_next_o_id` y `d_id` obtenidos en los pasos anteriores.

10. Se actualiza el campo `o_ol_cnt` de la fila de la tabla `orderr` cuyos campos `o_id` y `o_d_id` son iguales a los valores `d_next_o_id` y `d_id` obtenidos en

los pasos anteriores, con el número de líneas de orden procesadas. El valor calculado se escribe en la memoria compartida.

11. Se calcula el valor del campo `total_amount` y se escribe en la memoria compartida.

12. Se confirma la transacción mediante `EXEC SQL COMMIT`.

Después de cada una de las sentencias se SQL que implementan los pasos anteriones, se comprueba si se ha producido un error analizando el campo `sqlca.sqlcode`. Si se ha producido, de cancela la transacción y se escribe en el fichero `tm_err.log` el código del error, la descripción contenida en el campo `sqlca.sqlerrm.sqlerrmc`, además de alguna información adicional que permita localizar el error.

Mensaje de transacción Payment

Al recibir este mensaje, el Monitor de Transacciones ejecuta una transacción Payment con los datos recibidos en el mensaje de transacción, siguiendo el perfil de la cláusula 2.5 del estándar TPC-C. La función que se encarga de la ejecución de la transacción es `trans_payment()`. A esta función se le pasa como parámetros la estructura de datos de transacción recibida y el puntero a la memoria compartida. A medida que se va ejecutando la transacción se escriben los resultados en la memoria compartida. La estructura de datos se describe en el apartado 3.3.1.5.

A continuación se detalla la implementación del perfil de la transacción Payment de la cláusula 2.5 del TPC-C.

1. Se recoge la fecha del sistema mediante la función `getfechahora()`.

2. Se extraen los datos de la estructura de mensaje de transacción.

3. Se selecciona la fila del almacén cuyo atributo `w_id` coincide con el valor obtenido en la estructura de mensaje de transacción, y se extraen el valor de los atributos `w_name`, `w_street_1`, `w_street_2, w_city, w_state` y `w_zip`. Los valores obtenidos se escriben en la memoria compartida.

4. Se selecciona la fila de la tabla `district` cuyos campo `d_w_id, d_id` coincidan con los obtenidos en la estructura de mensaje y se extraen los valores `d_name, d_street_1, d_street_2, d_city, d_state` y `d_zip`. Los datos obtenidos se escriben en la memoria compartida.

5. Se actualiza en campo `d_ytd` sumando la cantidad `h_amount` recibida en el mensaje de transacción, de la fila de la tabla `district` cuyos campos `d_id, d_w_id` coincidan con los de la estructura de mensaje de transacción.

6. Se comprueba si el cliente ha sido seleccionado por `c_last`. Para ello si el campo `c_id` de la estructura de mensaje recibida contiene el valor 0, se considera que el cliente ha sido seleccionado por `c_last`.

Si ha sido seleccionado por `c_last`.

a) Se crea un cursor para la siguiente consulta: seleccionar las filas cuyos campos `c_w_id, c_d_id` y `c_last` coincidan con los datos recibidos en el mensaje de transacción, y recuperar los campos `c_id`, `c_first`, `c_middle`, `c_street_1`, `c_street_2`, `c_city`, `c_state`, `c_zip`, `c_phone`, `c_credit`, `c_credit_lim`, `c_discount`, `c_balance` y `c_since` ordenados en orden ascendente del campo `c_first`.

b) Se cuenta el número de filas de la tabla `customer` que cumplen la anterior condición.

c) Se sitúa el cursor en la posición intermedia de las coincidencias.

d) Se recuperan los datos solicitados en la consulta del cursor, y se escriben en la memoria compartida

7. Si no ha sido seleccionado por `c_last` se habrá seleccionado `c_id`, y se realizan los siguientes pasos:

   a) Se selecciona las fila de la tabla `customer` cuyos campos `c_w_id, c_d_id` y `c_id` coincidan con los datos recibidos en el mensaje de transacción, y recuperar los campos `c_last`, `c_fisrst`, `c_middle`, `c_street_1`, `c_street_2`, `c_city`, `c_state`, `c_zip`, `c_phone`, `c_credit`, `c_credit_lim`, `c_discount`, `c_balance` y `c_since`. Los valores obtenidos se escriben en la memoria compartida.

8. Se actualiza `c_balance` y `c_ytd_payment` restando y sumando respectivamente el valor de `h_amount` obtenido de la estructura de mensajes, de la fila cuyos campos `c_w_id`, `c_d_id` y `c_id` coinciden con los valores obtenidos de la estructura de mensaje.

9. Si el valor del atributo `c_credit` obtenido anteriormente contiene la cadena 'BC' se modifica al campo `c_data`, como se especifica en la cláusula 2.5.2.2 del TPC-C.

10. se inserta una nueva fila en la tabla `history` con los siguientes valores:

    - `h_c_d_id` se le asigna el valor del campo `c_d_id` de la estructura de mensaje.
    - `h_c_w_id` se le asigna el valor del campo `c_w_id` de la estructura de mensaje.
    - `h_c_id` se le asigna el valor del campo `c_id` obtenido anteriormente.
    - `h_d_id` se le asigna el valor del campo `d_id` de la estructura de mensaje.
    - `h_w_id` se le asigna el valor del campo `w_id` de la estructura de mensaje.
    - `h_date` se le asigna el valor de la fecha y la hora obtenido al inicio de la transacción.

- `h_amount` se le asigna el valor del campo `h_amount` de la estructura de mensaje.
- `h_data` se insertan los valores de los campos `w_name` y `d_name` obtenidos anteriormente.

11. Se confirma la transacción mediante `EXEC SQL COMMIT`.

Después de cada una de las sentencias se SQL que implementan los pasos anteriores, se comprueba si se ha producido un error analizando el campo `sqlca.sqlcode`. Si se ha producido, de cancela la transacción y se escribe en el fichero `tm_err.log` el código del error, la descripción contenida en el campo `sqlca.sqlerrm.sqlerrmc`, además de alguna información adicional que permita localizar el error.

Mensaje de transacción Order-Status

Al recibir este mensaje, el Monitor de Transacciones ejecuta una transacción Order-Status con los datos recibidos en el mensaje, siguiendo el perfil de la cláusula 2.6 del estándar TPC-C. La función que se encarga de la ejecución de la transacción es `trans_ostatus()`. A esta función se le pasa como parámetros la estructura de datos de transacción recibida y el puntero a la memoria compartida. La estructura de datos se describe en el apartado 2.3.1.5.

A continuación se detalla la implementación del perfil de la transacción Order-Status, de la cláusula 2.6 del TPC-C.

1. Se extraen los datos del mensaje de transacción.
2. Se comprueba si el cliente ha sido seleccionado por `c_id`, comprobando si el campo `c_id` no contiene un 0.
3. Si se ha seleccionado por `c_id` se seleccionan de la tabla `customer` los atributos `c_balance c_middle c_first c_last`, de la filas cuyos campos `c_w_id`, `c_d_id` y `c_id`, coincidan con los valores obtenidos de la estructura de mensaje.
4. Si el cliente ha sido seleccionado por `c_last` se realizan los siguientes pasos:
   a) Se crea un cursor, que realiza la siguiente consulta: seleccionar de la tabla `customer` los atributos `c_balance c_middle c_first` y `c_id`, de la filas cuyos campos `c_w_id`, `c_d_id` y `c_last`, coincidan con los valores obtenidos de la estructura de mensaje, ordenador por orden ascendente de `c_last`.
   b) Se cuenta el número de filas que cumplen la condición anterior.
   c) Se sitúa el cursor en la posición intermedia de las coincidencias.
   d) Se recuperan los datos solicitados en la consulta del cursor.

5. Los datos obtenidos se escriben en la memoria compartida.

6. Se declara otro cursor para hacer la siguiente consulta: seleccionar los campos `o_entry_d`, `o_id` y `o_carrier`, de las filas de la tabla `orderr` cuyos atributos `o_w_id`, `o_d_id` y `o_c_id` coinciden con los datos de la estructura de mensaje, en orden descendente del campo `o_id`.

7. Se obtienen los datos de la primera posición del cursor y se escriben en la memoria compartida.

8. Se declara otro cursor para hacer la siguiente consulta: seleccionar de la tabla `order_line` los capos `ol_i_id`, `ol_supply_w_id`, `ol_cuantity`, `ol_amount` y `ol_delivery_d`, de las filas cuyos campos `ol_w_id`, `ol_d_id` y `ol_o_id` coinciden con los valores de los campos `w_id` y `d_id` provenientes de la estructura de mensaje, y con el valor `o_id` obtenido anteriormente.

9. Para cada una de las posiciones del cursor, se escriben los datos obtenidos en la posición correspondiente del vector `item` de la estructura de mensaje de respuesta de la memoria compartida.

10. Se confirma la transacción.

Después de cada una de las sentencias de SQL que implementan los pasos anteriores, se comprueba si se ha producido un error analizando el campo `sqlca.sqlcode`. Si se ha producido, de cancela la transacción y se escribe en el fichero `tm_err.log` el código del error, la descripción contenida en el campo `sqlca.sqlerrm.sqlerrmc`, además de alguna información adicional que permita localizar el error.

Mensaje de transacción Delivery

Al recibir este mensaje, el Monitor de Transacciones ejecuta una transacción Delivery con los datos recibidos en el mensaje, siguiendo el perfil de la cláusula 2.7 del estándar TPC-C. La función que se encarga de la ejecución de la transacción es `trans_delivery()`. A esta función se le pasa como parámetros la estructura de datos de transacción recibida. La estructura de datos se describe en el apartado 2.3.1.5. A medida que se va ejecutando la transacción se escriben los resultados en el fichero `tm_delivery_res.log`.

A continuación se detalla la implementación del perfil de la transacción Delivery, de la cláusula 2.7 del TPC-C.

1. Se extraen los datos del mensaje de transacción.

2. Se escribe el sello de hora de encolado, el número de almacén y identificador de transportista que provienen de la estructura de mensaje de transacción.

3. Para cada uno de los distritos del almacén seleccionado se realizan las siguientes operaciones.

a) Se selecciona de la tabla new_order el menor valor del atributo `no_o_id`, de las filas cuyo campo `no_w_id` coincide con el valor del almacén obtenido en la estructura de mensaje y el campo `no_d_id` coincide con el número de distrito que se está procesando.

b) Si tras el paso anterior se produce un error y la estructura `sqlca` indica que no ha habido ninguna coincidencia, se pasa al distrito siguiente y se registra este caso en el fichero de resultados. En caso contrario se continúa.

c) Se elimina la anterior fila seleccionada de la tabla new_order.

d) Se selecciona de la tabla `orderr` el atributo `o_c_id` de las fila cuyos campos `o_w_id` coincide con obtenido en la estructura de mensaje, `o_d_id` coincide con el valor del distrito que se está procesando y `o_id` coincide con el valor del atributo `no_o_id` de la fila eliminada en el paso anterior.

e) En la tabla `orderr` se actualiza el atributo `o_carrier_id` con el valor obtenido en el mensaje de transacción, de la misma fila seleccionada en el paso anterior.

f) Se actualiza de la tabla `order_line` el campo `o_delivery_d` con el valor de la fecha actual del sistema, de todas las filas cuyos campos `ol_o_id` coincide con el valor del campo `no_o_id` obtenido anteriormente, `ol_w_id` coincide con el valor obtenido del mensaje de transacción y `ol_d_id` coincide con el distrito que se está procesando.

g) Se selecciona la suma de todos los valores `ol_amount` de la tabla `order_line` de las filas que cumplan la condición anterior.

h) Se actualiza en la tabla `customer` el campo `c_balance` con la suma obtenida en el paso anterior y el campo `c_delivery_cnt` se incrementa en una unidad, de la fila cuyos campos `c_w_id` coincide con el valor obtenido en el mensaje de transación, `c_d_id` coincide con el distrito que se está procesando y el campo `c_id` coincide con el valor de `o_c_id` obtenido anteriormente.

i) Se registra en el fichero de resultados el reparto para el distrito.

4. Se confirma la transacción.

Después de finalizar la transacción el MT toma un sello de hora que indica su finalización, y que junto con el sello de hora que ha recibido por la estructura de mensaje de transacción, permitirá calcular en el recuento de resultados el tiempo de ejecución de la transacción Delivery. El MT escribe una nueva línea en el fichero `tm_delivery_tr.log` con el número de almacén y de distrito del terminal que envió la transacción y los dos sellos de hora. En el fichero `tm_delivery_res.log` también se escribe el sello de hora de finalización de transacción.

Después de cada una de las sentencias se SQL que implementan los pasos anteriores, se comprueba si se ha producido un error analizando el campo `sqlca.sqlcode`. Si se ha producido, de cancela la transacción y se escribe en el fichero `tm_err.log` el código del error, la descripción contenida en el campo `sqlca.sqlerrm.sqlerrmc`, además de alguna información adicional que permita localizar el error.

Mensaje de transacción Stock-Level

Al recibir este mensaje, el Monitor de Transacciones ejecuta una Transacción Stock-Level con los datos recibidos en el mensaje, siguiendo el perfil de la cláusula 2.8 del estándar TPC-C. La función que se encarga de la ejecución de la transacción es `trans_stock_level()`. A esta función se le pasa como parámetros la estructura de datos de transacción recibida y el puntero a la memoria compartida. La estructura de datos se describe en el apartado 3.3.1.5.

A continuación se detalla la implementación del perfil de la transacción Stock-Level, de la cláusula 2.8 del TPC-C.

- Se extraen los datos del mensaje de transacción.
- Se selecciona de la tabla `district` el campo `d_next_o_id` de la fila cuyo valor de los campos `d_id` y `d_w_id` coinciden con los datos obtenidos de la estructura de mensaje.
- En la tabla `stock`, se realiza un recuento de todos los artículos de las filas de la tabla `stock` que cumplan:
  - que su campo `s_w_id` sea igual al valor del almacén obtenido en el mensaje de transacción.
  - el campo `s_i_id` aparezca en el campo `ol_i_id` de las filas de la tabla `order_line` que cumplan que `ol_w_id` y `ol_d_id` coincidan con los valores extraídos del mensaje de transacción, y que `ol_o_id` sea menor que `d_next_o_id` y mayor o igual que `d_next_o_id` menos 20 unidades (`d_next_o_id` se obtiene en el paso 1).
  - El campo `s_quantity` sea menor que el valor `threshold` obtenido del mensaje de transacción.

Después de cada una de las sentencias se SQL que implementan los pasos anteriores, se comprueba si se ha producido un error analizando el campo `sqlca.sqlcode`. Si se ha producido, de cancela la transacción y se escribe en el fichero `tm_err.log` el código del error, la descripción contenida en el campo `sqlca.sqlerrm.sqlerrmc`, además de alguna información adicional que permita localizar el error.

2.3.4.5 Etapa de Finalización

El Monitor de Transacciones inicia el proceso de finalización de su ejecución al recibir la señal `SIGTERM` enviada por el Controlador del Benchmark. Los pasos para la finalización de la ejecución son: eliminación de la cola de mensajes y desconexión de postgreSQL. Los mecanismos para efectuar estas operaciones se detallan en los apartados 2.3.1.5 y 2.3.1.1 respectivamente.

Como se explicó en el apartado dedicado al Modulo Controlador del Benchmark, el control de la finalización del test se realiza de tal manera que el monitor recibe la señal de terminación cuando todos los terminales se han desconectado del Monitor de Transacciones y han finalizado su ejecución. De esta forma, al llegar a la etapa de finalización, no hay ningún mensaje pendiente en la cola.

2.3.5. Controlador de Checkpoints

Este módulo se encarga de realizar los checkpoints según las especificaciones de la cláusula 5.5.2.2 del estándar TPC-C.

Se activa, al comienzo del intervalo de medida. Realiza checkpoints periódicos, cada 30 minutos, hasta recibir la señal `SIGTERM`, enviada por el controlador del benchmark al finalizar el test. La señal `SIGTERM` se enmascara con la función `sig_term()` que se encarga de modificar un flag que indica al Controlador de Checkpoints si ha de continuar o no realizando checkpoints.

Por cada checkpoint realizado, se registra una entrada en el fichero `check.log` con los sellos de hora de comienzo y de fin de cada checkpoint.

El esquema de funcionamiento del Controlador de Checkpoints se ilustra en la figura 2.15. A continuación se detallan cada una de las etapas:

A continuación se detallan cada una de las etapas.

Conexión

Se conecta con la base de datos tpcc mediante el procedimiento indicado en el apartado 3.3.1.1.

Tomar sello de hora de inicio

Se recoge la fecha del sistema.

Realizar un checkpoint

Se realiza un checkpoint mediante la sentencia `EXEC SQL CHECKPOINT`.

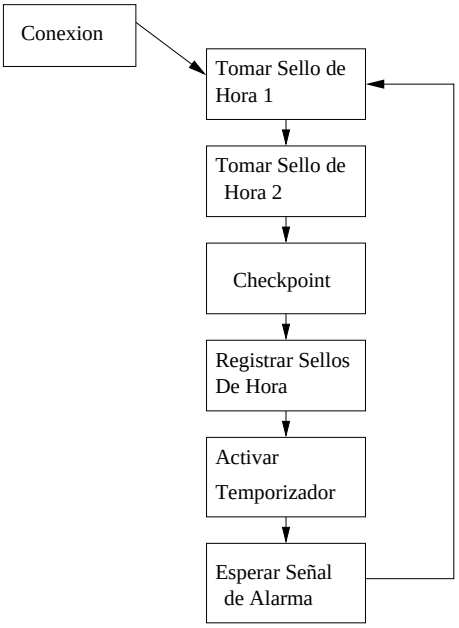


Figura 2.15: Funcionamiento del Cotrolador de Checkpoints

Tomar sello de hora de fin

Este paso es análogo al del sello de hora de inicio.

Registrar Sellos de Hora

Se escriben los sellos de hora obtenidos anteriormente en el fichero `check.log`.

Activar el Temporizador

Se inicia un temporizador de 30 minutos mediante la función `alarm()` de UNIX.

Esperar a señal `SIGALRM`

Cuando el temporizador llega a 0 el sistema envía la señal `SIGALRM`. al Controlador provocando que este deje de esperar. Si en este punto no se ha recibido la señal `SIGTERM` proveniente del Controlador del Benchmark, se retorna a la etapa de Tomar sello de hora de inicio. De lo contrario de finaliza la ejecución.

2.3.6. Controlador de Limpiezas

Se inicia al comienzo del test si el usuario decide realizar limpiezas en la base de datos durante el test. Ejecuta periódicamente el comando `EXEC SQL VACUUM ANALYZE` [27] de postgreSQL, que elimina de la base de datos la información residual resultante de la ejecución de transacciones y analiza las tablas de estadísticas

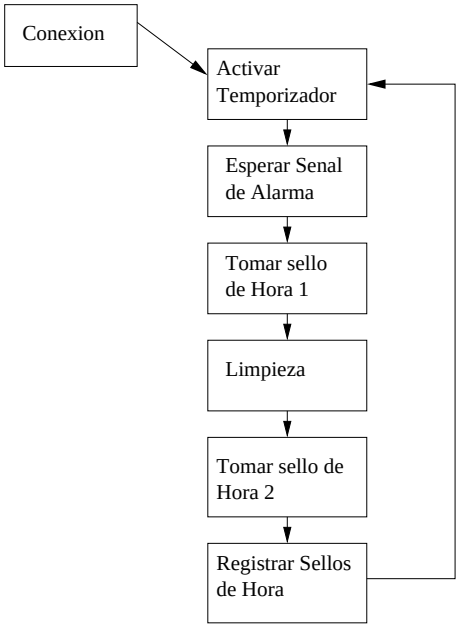


Figura 2.16: Funcionamiento del Cotrolador de Limpiezas

utilizadas por el optimizador de consultas para mejorar su velocidad. El periodo entre cada limpieza y el número total de limpiezas se pasan como parámetros a este módulo. Si como número máximo se le pasa un cero, realizará limpiezas hasta que el Controlador del Benchmark le indique que ha finalizado el test mediante la señal `SIGTERM`.

El esquema de funcionamiento del Controlador de Limpiezas se ilustra en la figura 2.16.

Se repiten todas las etapas hasta recibir la señal `SIGTERM` o haber realizado el número de check points que se ha pasado como parámetro.

Conexión

Se conecta con la base de datos `tpcc`.

Activar el Temporizador

Se arranca un temporizador con el periodo que se ha pasado como parámetro mdiante la función `alarm`.

Esperar a señal `SIGALRM`

Cuando el temporizador llega a 0, el sistema envía la señal `SIGALRM` al Controlador de Limpiezas, que captura esa señal y pasa a la siguiente etapa.

Tomar sello de hora de inicio

Se recoge la fecha del sistema.

Realizar una limpieza

Se ejecuta una limpieza mediante la sentencia `EXEC SQL VACUUM ANALIZE`.

Tomar Sello de Hora de Fin

Este paso es análogo al de Tomar Sello de Hora de Inicio.

Registrar Sellos de Hora

Se escriben los sellos de hora obtenidos anteriormente en el fichero `vacuum.log`.

Tras esto si no se han ejecutado el número de limpiezas indicado en la llamada se retorna al paso de activación del temporizador. De lo contrario se espera a que el Controlador de Benchmark envíe la señal `SIGTERM`, indicando la finalización del test. La señal `SIGTERM` se enmascara con la función `sig_term()`, que se encarga de modificar en flag que indica al Controlador que debe seguir realizando limpiezas o que hace que este espere a la señal `SIGTERM` cuando ha realizado el número de limpiezas que se le ha indicado.

Si el número máximo de limpiezas que se pasó como parámetro es cero, siempre se retorna a la activación del temporizador.

2.3.7. Funciones del benchmark

En este apartado se describen las funciones de que se compone cada módulo del benchmark, así como las funciones comunes a varios módulos que están declaradas en el fichero de cabecera `tpcc.h`.

Funciones del módulo Controlador del Benchmark

Las Funciones que están declaradas en el Controlador del Benchmark son:

```
int lanzador(int w, int term)
```

Recibe como parámetro el número de almacenes `w` y el número de terminales por almacén `term` con que se va a ejecutar el test.

Se encarga de lanzar el Monitor de Transacciones y población de Emuladores de Terminal Remoto para la realización del test.

Lanza *term* terminales por cada *w* almacenes.

Reserva memoria para almacenar el *pid* del Monitor de Transacciones y lo apunta con un puntero global de tipo `pid_t`.

Reserva memoria para almacenar en un vector todos los *pid* de los Terminales y la apunta con un puntero global.

Devuelve 1 si se han lanzado todos los módulos sin problemas y 0 en caso contrario.

```
int lanza_check()
```

No recibe parámetros.

Se encarga de lanzar el Controlador de Checkpoints.

Reserva memoria para almacenar el *pid* del Controlador de Checkpoints, y lo apunta con un puntero global de tipo `pid_t`.

Devuelve 1 si se ha lanzado el controlador sin problemas y 0 en caso contrario.

```
int lanza_vacuum(int intv, int num)
```

Recibe como parámetros el intervalo entre limpiezas `intv` y el número máximo de estas `num`. Se encarga de lanzar el Controlador de Limpiezas, pasándole como parámetros `intv` y `num`.

Reserva memoria para almacenar el `pid` del Controlador de Limpiezas, y la apunta con un puntero global de tipo `pid_t`.

Devuelve 1 si se ha lanzado el controlador sin problemas y 0 en caso contrario.

```
void restaura()
```

No recibe parámetros.

Se encarga de restaurar las modificaciones producidas por las transacciones New-Order, Payment y Delivery en la base de datos resultantes de un test anterior. Para ello, dentro de esta función se llama a las funciones `restaura_new_order()`, `restaura_payment()` y `restaura_delivery()`, a las que les pasa el número de almacenes que contiene la base de datos.

```
void restaura_new_order(long num_alma)
```

Recibe como parámetro el número de almacenes que contiene la base de datos.

Se encarga de deshacer las modificaciones realizadas por la transacción New-Order en un test anterior.

```
void restaura_payment(long num_alma)
```

Recibe como parámetro el número de almacenes que contiene la base de datos.

Se encarga de deshacer las modificaciones realizadas por la transacción Payment en un test anterior.

```
void restaura_delivery(long num_alma)
```

Recibe como parámetro el número de almacenes que contiene la base de datos.

Se encarga de deshacer las modificaciones realizadas por la transacción Delivery en un test anterior.

```
void consistencia()
```

No recibe parámetros.

Se encarga de comprobar que la base de datos `tpcc` cumple las condiciones de consistencia definidas en las cláusulas 3.3.2.1, 3.3.2.2, 3.3.2.3, y 3.3.2.4 del estándar TPC-C. Para ello dentro de esta función se llama a las funciones `condicion_1()`, `condicion_2()`, `condición_3()` y `condición_4()`, a las que les pasa el número de almacenes que contiene la base de datos.

```
void condicion_1(long num_alm)
```

Recibe como parámetro el número de almacenes que contiene la base de datos.

Comprueba que la base de datos cumple con la condición de consistencia definida en la cláusula 3.3.2.1 del estándar TPC-C.

```
void condicion_2(long num_alm)
```

Recibe como parámetro el número de almacenes que contiene la base de datos.

Comprueba que la base de datos cumple con la condición de consistencia definida en la cláusula 3.3.2.2 del estándar TPC-C.

```
void condicion_3(long num_alm)
```

Recibe como parámetro el número de almacenes que contiene la base de datos.

Comprueba que la base de datos cumple con la condición de consistencia definida en la cláusula 3.3.2.3 del estándar TPC-C.

```
void condicion_4(long num_alm)
```

Recibe como parámetro el número de almacenes que contiene la base de datos.

Comprueba que la base de datos cumple con la condición de consistencia definida en la cláusula 3.3.2.4 del estándar TPC-C.

`int carga (int w)`

Recibe como parámetro el número de almacenes con que se quiere crear la base de datos.

Se encarga de crear una base de datos `tpcc` válida para la realización del test. Se crean las nueve tablas según las especificaciones de la cláusula 1, y se pueblan según las especificaciones de la cláusula 4.

Dentro de esta función se llama a la función `creartablas()` que crea las nueve tablas de la base de datos.

Para la población de las tablas se llama a las funciones: `poblar_warehouse()`, `poblar_item()`, `poblar_district()`, `poblar_customer()`, `poblar_stock()`, `poblar_history()`, `poblar_order()` y `poblar_new_order()`.

En esta función se generan las constantes aleatorias necesarias para la población de la base de datos y se almacenan en el fichero `cons.dat`.

Retorna -1 si no se ha podido crear la base de datos y 1 en caso contrario.

`void creartablas()`

No recibe parámetros.

Se encarga de la creación de las nueve tablas de la base de datos según las especificaciones de la cláusula 1 del estándar TPC-C.

`void poblar_warehouse(int w)`

Recibe como parámetro el número de almacenes con que se quiere crear la base de datos.

Se encarga de poblar la tabla `warehouse` según las especificaciones de la cláusula 4.

`void poblar_district(int w)`

Recibe como parámetro el número de almacenes con que se quiere crear la base de datos.

Se encarga de poblar la tabla `district` según las especificaciones de la cláusula 4.

`void poblar_item()`

No recibe parámetros.

Se encarga de poblar la tabla `item` según las especificaciones de la cláusula 4.

`void poblar_stock(int n)`

Recibe como parámetro el número de almacenes con que se quiere crear la base de datos.

Se encarga de poblar la tabla `stock` según las especificaciones de la cláusula 4.

`void poblar_customer(int w)`

Recibe como parámetro el número de almacenes con que se quiere crear la base de datos.

Se encarga de poblar la tabla `customer` según las especificaciones de la cláusula 4.

`void poblar_history(int w)`

Recibe como parámetro el número de almacenes con que se quiere crear la base de datos.

Se encarga de poblar la tabla `history` según las especificaciones de la cláusula 4.

`void poblar_order(int w)`

Recibe como parámetro el número de almacenes con que se quiere crear la base de datos.

Se encarga de poblar las tablas `orderr` y `order_line` según las especificaciones de la cláusula 4.

Por cada fila de la tabla `orderr` se inserta un número aleatorio de filas en la tabla `order_line` comprendido entre 5 y 15 con una media de 10.

`void poblar_new_order(int w)`

Recibe como parámetro el número de almacenes con que se quiere crear la base de datos.

Se encarga de poblar la tabla `new_order` según las especificaciones de la cláusula 4.

`int menu(int haybd, int haylogs)`

Recibe los parámetro `haydb` y `haylogs`.

Muestra por pantalla opciones de menú que el usuario puede realizar en función del valor de esos parámetros.

Recoge la opción seleccionada por el usuario.

Retorna la opción elegida por el usuario.

`void zero(char *v, long tam)`

Recibe como parámetro un puntero a un caracter, y el parámetro `tam`.

Pone a cero `tam` bytes a partir de la dirección apuntada por `v`.

`int lectura_bitacora()`

No recibe parámetros.

Se encarga se hacer el recuento de los resultados del test a partir de la bitácoras que escriben el Monitor de Transacciones y la población de Emuladores de Terminal Remoto.

Muestra por pantalla esos rendimientos.

Genera el fichero donde se almacenan los resultados de rendimiento en un fichero cuyo nombre lo elige el usuario.

Genera los ficheros de datos necesarios para la realización de las gráficas, que se especifican en el estándar TPC-C. Los nombres de estos ficheros son: 'g1New_Order.dat', 'g1Delivery.dat', 'g1Payment.dat', 'g1OrderStatus.dat', 'g1StockLevel.dat', 'g3.dat' y 'g4.dat'.

También se generan los ficheros 'vacuum.dat' y 'check.dat' donde se muestra información de las limpiezas y de los checkpoints realizados durante el test.

Retorna 1 en caso de que se hayan realizado todas las operaciones correctamente, y 0 en caso de que no se hayan podido leer las bitácoras.

`int hay_tablas(int w, int caso)`

Recibe como parámetros el número de almacenes con que está poblada la base de datos y el entero `caso`.

Se encarga de comprobar que las tablas de la base de datos poseen la cardinalidad mínima necesaria en cada '`caso`'. Este parámetro determina el caso de comprobación:

Los casos posibles de comprobación son para la realización del test (`caso=0`), y para la restauración de la base de datos (`caso=1`).

Si `caso=0` se comprueba la cardinalidad de las tablas, para asegurar que existe el número mínimo necesario para el test, pero solo se examinan las filas que

dependan de los almacenes indicados en 'w'.

Si `caso=1` se comprueba la cardinalidad de todas las tablas, menos de la tabla new_order, para asegurar que existe el número mínimo necesario para realizar la restauración. No se comprueba la tabla new_order ya que va a ser poblada de nuevo completamente en la restauración.

Retorna 1 si se cumplen los requisitos y 0 en caso contrario.

```
void estado_base()
```

No recibe parámetros.

Muestra por pantalla la cardinalidad de las nueve tablas de la base de datos.

```
int hay_fich(char *punt)
```

Recibe como parámetro un puntero a caracter que apunta al nombre de un fichero.

Comprueba la existencia de ese fichero retornando 1 en caso de existir y 0 en caso contrario.

```
void signal_int1()
```

No recibe parámetros.

Esta función enmascara a la señal `SIGINT` en las etapas de control de la restauración y de la carga. Cuando se recibe esta señal se interrumpe la ejecución de la etapa correspondiente, enviando la señal `SIGSTOP` al proceso encargado de ejecutarla. Tras esto se pregunta el usuario si desea terminar con el proceso. En caso afirmativo envía la señal `SIGTERM` al proceso correspondiente, para que finalice su ejecución. Si el usuario decide no terminar con el proceso se envía la señal `SIGCONT` que provoca que el proceso interrumpido continúe.

```
void signal_int2()
```

No recibe parámetros.

Esta función evita el efecto producido por la señal `SIGINT` en las etapas de carga de la base de datos, restauración de la base de datos, comprobación de la consistencia y recuento de resultados.

```
void signal_int3()
```

No recibe parámetros.

Esta función enmascara a la señal `SIGINT` en la etapa de control del test.

Cuando se recibe esta señal pregunta el usuario si desea terminar con el proceso. En caso afirmativo se fuerza al valor '1' el flag global *terminado* que controla la ejecución del test, provocando la desactivación de los procesos que intervienen en él. Si el usuario decide continuar con el test, el valor del flag no se modifica.

`void signal_usr1()`

No recibe parámetros.

Esta función enmascara a la señal `SIGUSR1` en las etapa de control de carga y de restauración. Esta señal la envían los procesos encargados de la ejecución de la etapa correspondiente para indicar a la etapa de control que han terminado su ejecución. Cuando se recibe esta señal se fuerza al valor '1' el flag global `terminado` que controla la ejecución de la etapa.

`void signal_alarm()`

No recibe parámetros.

Enmascara a la función `SIGALRM` que envía el sistema operativo cuando el temporizador activado en la etapa de test llega a cero. Cuando se recibe esta señal se fuerza al valor '1' el flag global `timeout` que controla los bucles de espera del Controlador del Test.

`void signal_term()`

No recibe parámetros.

Enmascara a la función `SIGTERM` en el Controlador del Benchmark. Cuando se recibe esta señal, independientemente de la etapa que se esté ejecutando, se ponen los medios para terminar la ejecución del programa. Para ello, se fuerzan al valor '1' el valor de los flags globales `teminado` y `salir` que controlan los bucles de espera y el bucle principal del del Controlador del Benchmark.

`void signal_term_exit()`

No recibe parámetros.

Enmascara a la función `SIGTERM` en las etapas de ejecución de la carga y la restauración de la base de datos. Cuando se recibe la señal se desconecta de postgreSQL y se termina la ejecución de la etapa correspondiente.

Funciones del Monitor de Transacciones

```
int trans_new_order(struct tnew_order_men *new_order,
union tshm *shm)
```

Recibe como parámetros un puntero a la estructura de mensaje de transacción New-Order, y un puntero a la memoria compartida.

Se encarga de implementar el perfil de transacción New-Order definido en la cláusula 2.4 del TPC-C con los datos contenidos en la estructura de mensajes. Escribe los resultados en la memoria compartida.

Registra los posibles errores resultantes de la ejecución de la transacción en el fichero `tm_err.log`.

La función devuelve el valor -1 si se ha cancelado la transacción a causa de un error y 0 en caso contrario. En caso de que se haya cancelado a causa de que la transacción contenía un artículo inválido, también se devuelve el valor 0.

```
int trans_payment(struct tpayment_men *payment, union tshm *shm)
```

Recibe como parámetros un puntero a la estructura de mensaje de transacción Payment, y un puntero a la memoria compartida.

Implementa el perfil de transacción Payment definido en la cláusula 2.5 del TPC-C con los datos contenidos en la estructura de mensajes. Escribe los resultados en la memoria compartida.

Registra los posibles errores resultantes de la ejecución de la transacción en el fichero `tm_err.log`.

Retorna el valor -1 si se ha producido un error durante la ejecución de la transacción, en caso contrario se devuelve 0.

```
int trans_ostatus(struct torder_status_men *ostatus, union tshm *shm)
```

Recibe como parámetros un puntero a la estructura de mensaje de transacción Order-Status, y un puntero a la memoria compartida.

Implementa el perfil de transacción Payment definido en la cláusula 2.6 del TPC-C con los datos contenidos en la estructura de mensajes. Escribe los resultados en la memoria compartida.

Registra los posibles errores resultantes de la ejecución de la transacción en el fichero `tm_err.log`.

Retorna el valor -1 si se ha producido un error durante la ejecución de la transacción, en caso contrario se devuelve 0.

```
int trans_delivery(struct tdelivery_men *delivery, union tshm *shm)
```

Recibe como parámetros un puntero a la estructura de mensaje de transacción Delivery.

Implementar el perfil de transacción Delivery definido en la cláusula 2.7 del TPC-C con los datos contenidos en la estructura de mensajes. Escribe los resultados en el fichero de resultados `tm_delivery_res.log`.

Registra los posibles errores resultantes de la ejecución de la transacción en el fichero `tm_err.log`.

```
int trans_stock_level(struct tstock_level_men *stock_level,
union tshm *shm)
```

Recibe como parámetros un puntero a la estructura de mensaje de transacción Stock-Level, y un puntero a la memoria compartida.

Implementa el perfil de transacción Stock-Level definido en la cláusula 2.8 del TPC-C con los datos contenidos en la estructura de mensajes. Escribe los resultados en la memoria compartida.

Registra los posibles errores resultantes de la ejecución de la transacción en el fichero `tm_err.log`.

Retorna el valor -1 si se ha producido un error durante la ejecución de la transacción, en caso contrario se devuelve 0.

```
void sigterm()
```

No recibe parámetros.

Esta función se ejecuta cuando se recibe la señal `SIGTERM`.

Modifica el flag global `salir` forzándolo al valor 1. El flag `salir` controla el bucle principal del módulo MT. Cuando toma el valor 1, el MT deja de atender solicitudes e interrumpe su ejecución.

```
void ctl_c()
```

No recibe parámetros.

Se invoca cuando se recibe la señal `SIGINT`.

No realiza ninguna operación. Su misión es desactivar el efecto de la señal `SIGINT` provocada por el *<ctrl-C>* que introduce el usuario.

Funciones del Emulador de Terminal Remoto

```
long aleat_tpensar(int media, char *estado)
```

Recibe como parámetros la media de la distribución y el puntero al vector de

estado para generar el tiempo de pensar.

Devuelve un número entero que corresponderá con el tiempo de pensar expresado en segundos. La generación de este número se realiza de acuerdo con el método definido en la cláusula 5.2.5.4 del TPC-C:

```
tp = -ln(r * tm)
```

donde: `r` es un número aleatorio uniformemente distribuido entre [0,1] calculado mediante la función `random()` y `tm` la media de la distribución.

```
void genera_datos_new_order(long w, struct tnew_order_men
new_order)
```

Recibe como parámetros el número de almacén `w` y la dirección de la estructura de transacción New-Order.

Genera los datos de la transacción New-Order y los introduce en la estructura de transacción. Cada dato generado se valida para simular los mecanismos de comprobación de campos de los terminales reales. Si un dato generado resulta incorrecto, se vuelve a generar.

Para validar los datos se utiliza la función: `es_entero()`.

```
void genera_datos_payment(long w, struct tpayment_men *payment)
```

Recibe como parámetros el número de almacén `w` y la dirección de la estructura de transacción Payment.

Genera los datos de la transacción Payment y los introduce en la estructura de transacción. Cada dato generado se valida para simular los mecanismos de comprobación de campos de los terminales reales. Si un dato generado resulta incorrecto, se vuelve a generar.

Para validar los datos se utilizan las funciones: `es_entero()`, `es_alfa()` y `es_real()`.

```
void genera_datos_order_status(long w, struct torder_status_men
ostatus)
```

Recibe como parámetros el número de almacén `w` y la dirección de la estructura de transacción Order-Status.

Genera los datos de la transacción Order-Status y los introduce en la estructura de transacción. Cada dato generado se valida para simular los mecanismos de comprobación de campos de los terminales reales. Si un dato generado resulta incorrecto, se vuelve a generar.

Para validar los datos se utilizan la funciones: `es_entero()` y `es_alfa()`.

`void genera_datos_stock_level(long w, struct tstock_level_men stock_level)`

Recibe como parámetros el número de almacén `w` y la dirección de la estructura de transacción Stock-Level.

Genera los datos de la transacción Stock-Level y los introduce en la estructura de transacción. Cada dato generado se valida para simular los mecanismos de comprobación de campos de los terminales reales. Si un dato generado resulta incorrecto, se vuelve a generar.

Para validar los datos se utiliza la función: `es_entero()`.

`void genera_datos_delivery(long w, struct tdelivery_men *delivery)`

Recibe como parámetros el número de almacén `w` y la dirección de la estructura de transacción Delivery.

Genera los datos de la transacción Delivery y los introduce en la estructura de transacción. Cada dato generado se valida para simular los mecanismos de comprobación de campos de los terminales reales. Si un dato generado resulta incorrecto, se vuelve a generar.

Para validar los datos se utiliza la función: `es_entero()`.

`void pant_new_order_pet()`

No recibe parámetros.

Muestra la pantalla de petición de datos de la transacción New-Order.

`void pant_new_order_muest(struct tnew_order_men *new_order)`

Recibe como parámetros la dirección de la estructura de transacción New-Order.

Muestra la pantalla de transacción New-Order con el valor de los campos de la estructura de transacción generados previamente mediante la función `genera_datos_new_order()`.

`void pant_new_order_menu(struct tnew_order_men *new_order)`

Recibe como parámetros la dirección de la estructura de transacción New-Order.

Muestra la pantalla de menú de la transacción New-Order. Se muestra la in-

formación resultante de la ejecución de la transacción New-Order contenida en la memoria compartida. También se muestran los datos generados previamente mediante la función `genera_datos_new_order()`.

`void pant_payment_pet()`

No recibe parámetros.

Muestra la pantalla de petición de datos de la transacción Payment.

`void pant_payment_muest(struct tpayment_men *payment)`

Recibe como parámetros la dirección de la estructura de transacción Payment.

Muestra la pantalla de transacción Payment con el valor de los campos de la estructura de transacción generados previamente mediante la función `genera_datos_payment()`.

`void pant_payment_menu(struct tpayment_men *payment)`

Recibe como parámetros la dirección de la estructura de transacción Payment.

Muestra la pantalla de menú de la transacción Payment. Se muestra la información resultante de la ejecución de la transacción Payment contenida en la memoria compartida. También se muestran los datos generados previamente mediante la función `genera_datos_payment()`.

`void pant_ostatus_pet()`

No recibe parámetros.

Muestra la pantalla de petición de datos de la transacción Order-Status.

`void pant_ostatus_muest(struct torder_status_men *ostatus)`

Recibe como parámetros la dirección de la estructura de transacción Order-Status.

Muestra la pantalla de transacción Order-Status con el valor de los campos de la estructura de transacción generados previamente mediante la función `genera_datos_order_status()`.

`void pant_ostatus_menu(struct torder_status_men *ostatus)`

Recibe como parámetros la dirección de la estructura de transacción Order-Status.

Muestra la pantalla de menú de la transacción Order-Status. Se muestra la información resultante de la ejecución de la transacción Order-Status contenida

en la memoria compartida. También se muestran los datos generados previamente mediante la función `genera_datos_order_status()`.

`void pant_delivery_pet()`

No recibe parámetros.

Muestra la pantalla de petición de datos de la transacción Delivery.

`void pant_delivery_muest(struct tdelivery_men *delivery)`

Recibe como parámetros la dirección de la estructura de transacción Delivery.

Muestra la pantalla de transacción Delivery con el valor de los campos de la estructura de transacción generados previamente mediante la función `genera_datos_delivery()`.

`void pant_delivery_menu(struct tdelivery_men *delivery)`

Recibe como parámetros la dirección de la estructura de transacción Delivery.

Muestra la pantalla de menú de la transacción Delivery. Se muestra la información resultante de la ejecución de la transacción Delivery contenida en la memoria compartida. También se muestran los datos generados previamente mediante la función `genera_datos_delivery()`.

`void pant_stock_level_pet()`

No recibe parámetros.

Muestra la pantalla de petición de datos de la transacción Delivery.

`void pant_stock_level_muest(struct tstock_level_men *stock_level)`

Recibe como parámetros la dirección de la estructura de transacción Stock-Level.

Muestra la pantalla de transacción Stock-Level con el valor de los campos de la estructura de transacción generados previamente mediante la función `genera_datos_stock_level()`.

`void pant_stock_level_menu(struct tstock_level_men *stock_level)`

Recibe como parámetros la dirección de la estructura de transacción Stock-Level.

Muestra la pantalla de menú de la transacción Stock-Level. Se muestra la información resultante de la ejecución de la transacción Stock-Level contenida en la memoria compartida. También se muestran los datos generados previamente

mediante la función `genera_datos_stock_level()`.

`void leyenda()`

No recibe parámetros.

Muestra por pantalla el modo de invocar al programa. Se ejecuta cuando el usuario introduce en la llamada al programa un número o tipo de parámetros erróneo.

`void sigterm()`

No recibe parámetros.

Se llama a esta función cuando se recibe la señal `SIGTERM`. Esta función fuerza a '1' el valor del flag que controla el bucle principal del programa, provocando que el ETR se desconecte tras completar la transacción en curso.

`void ctl_c()`

No recibe parámetros.

Se llama a esta función cuando se recibe la señal `SIGINT`. No se realiza ninguna operación sino que únicamente evita el efecto de la señal.

Funciones del Controlador de Checkpoints

`void_alarma()`

No recibe parámetros.

Esta función se invoca cuando se recibe la señal `SIGALRM`. Fuerza a '1' el valor del flag global `alarma` que controla el bucle utilizado para esperar el intervalo entre checkpoints.

`void sig_int()`

No recibe parámetros.

Se llama a esta función cuando se recibe la señal `SIGINT`. No se realiza ninguna operación sino que únicamente se evita el efecto de la señal.

`void sig_term()`

No recibe parámetros.

Esta función se invoca cuando se recibe la señal `SIGTERM`. Fuerza a '0' el valor

del flag global `no_term` que controla el bucle principal del programa. Esto provoca que el programa detenga su ejecución.

Funciones del Controlador de Limpiezas

`void leyenda()`

No recibe parámetros.

Muestra por pantalla el modo de invocar al programa. Se ejecuta cuando el usuario introduce en la llamada al programa un número o tipo de parámetros erróneo.

`void sig_alarma()`

No recibe parámetros.

Esta función se invoca cuando se recibe la señal `SIGALRM`. Fuerza a '1' el valor del flag global `alarma` que controla el bucle utilizado para esperar el intervalo entre limpiezas.

`void sig_int()`

No recibe parámetros.

Se llama a esta función cuando se recibe la señal `SIGINT`. No se realiza ninguna operación sino que únicamente se evita el efecto de la señal.

`void sig_term()`

No recibe parámetros.

Esta función se invoca cuando se recibe la señal `SIGTERM`. Fuerza a '0' el valor del flag global `no_term` que controla el bucle principal del programa. Esto provoca que el programa detenga su ejecución.

Funciones Definidas en el Fichero de Cabecera `tpcc.h`

`long trunc(double f)`

Recibe como parámetro un número real de tipo `double`. Devuelve un entero de tipo `long` con la parte entera del valor recibido.

`long decimal(double f, int pos)`

Recibe como parámetros un número real de tipo `double` y un entero de tipo `int`. Devuelve un entero de tipo `long` con las 'pos' primeras cifras de la parte decimal del valor recibido.

```
double aleat_dbl(double inicio, double fin)
```

Recibe como parámetros dos números reales de tipo `double`. Genera un número aleatorio real de tipo double con distribución uniforme entre [`inicio`,`fin`].

Esta función utiliza la función `random()` de C.

```
long aleat_int(long inicio, long fin)
```

Recibe como parámetros dos números reales de tipo `long`. Genera un número aleatorio entero de tipo long con distribución uniforme entre [inicio,fin].

Esta función utiliza la función `random()` de C.

```
char *getfechahora(char *cad)
```

Recibe como parámetro la dirección del primer elemento de una cadena de caracteres. Devuelve la dirección de la cadena con la fecha y la hora actual del sistema.

Esta función ululiza las funciónes de C: `time()` y `localtime()`.

```
char *ctime_sin_salto(time_t *tiempo)
```

Recibe como parámetro un puntero de tipo `time_t` que representa el tiempo de calendario y lo convierte a una cadena de la forma:

"Wed Jun 30 21:49:08 1993"

La función retorna el puntero a la cadena.

En esta función se utiliza la función de C `c_time`.

```
int cad_alfa_num(int ini, int fin, char *cadena, char *estado1,
char *estado2)
```

Genera una cadena de caracteres alfanuméricos de tamaño comprendido entre [`ini`,`fin`].

Los parámetros que toma la función son:

- `ini` y `fin`: dos enteros de tipo `int` que indican el intervalo de la longitud de la cadena.
- `cadena`: la dirección de la cadena.

- `estado1`: puntero al vector de estado para generar la longitud de la cadena.
- `estado2`: puntero al vector de estado para generar los caracteres alfanuméricos.

La función retorna el tamaño de la cadena generada.

```
int cad_num(int tam, char *cadena, char *estado)
```

Genera una cadena de caracteres numéricos de tamaño `tam`.

Los parámetros que toma la función son:

- `tam`: tamaño de la cadena.
- `cadena`: la dirección de la cadena.
- `estado`: puntero al vector de estado para generar los caracteres numéricos.

La función retorna el tamaño de la cadena generada.

```
int crea_clast(int num, char *cadena)
```

Genera el nombre del cliente según se especifica en la cláusula 4.3.2.3 del tpc-c.

Los parámetros que toma la función son:

- `num`: número a partir del cual se genera el nombre del cliente.
- `cadena`: dirección de la cadena a generar.

La función retorna el tamaño de la cadena generada.

```
long nurand(long A, long x, long y, long C, char *estado1,
char *estado2)
```

Genera el un número aleatorio con distribución no uniforme siguiendo el método descrito en la cláusula 2.1.6 del TPC-C.

Los parámetros que toma la función son:

- `A`: constante aleatoria A.
- `x`,`y`: rango de valores en que se genera el número aleatorio.
- `C`: constante aleatoria C.

- `estado1`: vector de estado para generar un número aleatorio entre 0 y A.
- `estado2`: vector de estado para generar un número aleatorio entre x e y.

La función retorna el número aleatorio generado.

Para generar los números aleatorios intermedios se utiliza la función `aleat_int()`.

```
void permutacion_int(int *v, int tam, char *estado)
```

Genera un vector de tamaño `tam` con una permutación de los números naturales entre 1 y `tam`.

Los parámetros que toma la función son:

- `v`: puntero al vector.
- `tam`: tamaño del vector.
- `estado`: puntero al vector de estado utilizado para generar la permutación.

Esta función hace uso de `aleat_int()`.

```
void posicion_cartas(int *v, int tam,char *estado)
```

Genera un vector de tamaño `tam` con una permutación de los números entre entre 0 y `tam-1`.

Los parámetros que toma la función son:

- `v`: puntero al vector.
- `tam`: tamaño del vector.
- `estado`: puntero al vector de estado utilizado para generar la permutación.

Esta función hace uso de `aleat_int()`.

```
long buscar_n_vector(long *v, long tam, long num)
```

Busca el entero `num` dentro del vector apuntado por `v`, utilizando el método de búsqueda binaria.

Los parámetros que toma la función son:

- `v`: puntero al vector.

- `tam`: tamaño del vector.
- `num`: número a buscar.

`void insert_ord(long *v, long tam, long num)`

La función inserta el entero `num` en la posición correspondiente dentro del vector de números ordenados en orden ascendente apuntado por `v`.

Los parámetros que toma la función son:

- `v`: puntero al vector.
- `tam`: tamaño del vector.
- `num`: número a insertar.

`void aleat_vect(long *v, long tam, long i, long f)`

Genera un vector de tamaño `tam` de números aleatorios no repetidos comprendidos entre [`i`,`f`] y ordenados en orden ascendente.

Los parámetros que toma la función son:

- `v`: puntero al vector.
- `tam`: tamaño del vector.
- `i`,`f`: intervalo en que se generan los números aleatorios.

Esta función llama a las funciones `aleat_int()` y `buscar_n_vector()`.

`double resta_tiempos(struct timeb *tAnt, struct timeb *tPost)`

Devuelve la diferencia de tiempo, expresada en segundos, entre los sellos de hora pasados como parámetro.

Los parámetros que toma la función son:

- `tAnt`: puntero a una estructura `timeb` que representa el sello de hora más antiguo.
- `tPost`: puntero a una estructura `timeb` que representa el sello de hora más reciente.

```
int es_entero(char *s)
```

Recibe como parámetro un puntero a una cadena.

Retorna 1 si la cadena representa un número entero, 0 en caso contrario.

```
int es_real(char *s)
```

Recibe como parámetro un puntero a una cadena.

Retorna 1 si la cadena representa un número real, 0 en caso contrario.

```
int es_alfa(char *s)
```

Recibe como parámetro un puntero a una cadena.

Retorna 1 si la cadena contiene únicamente caracteres alfanuméricos, 0 en caso contrario.

En este capítulo se han expuesto todas las consideraciones acerca del desarrollo y la implementación del benchmark TPCC-UVA. En el siguiente se explicará la forma de ponerlo en funcionamiento así como su modo de uso.

Capítulo 3

Manual de usuario

En este capítulo se explicarán los pasos necesarios para poner en marcha el software del benchmark TPCC-UVA, y se dará una descripción detallada de su modo de uso.

Se considera que en este punto el usuario conoce el estándar TPC-C, por lo que únicamente se explicarán los aspectos imprescindibles para la medición del rendimiento de la máquina puesta a prueba.

3.1. Instalación y configuración del entorno

El buen funcionamiento del benchmark depende en gran medida de la correcta instalación y configuración del entorno, por lo que se recomienda seguir rigurosamente los pasos de instalación que se detallan a continuación.

3.1.1. Instalación del motor de base de datos postgreSQL

La versión de postgreSQL utilizada para el desarrollo del benchmark es la 7.1.3. A pesar de que las versiones posteriores son compatibles, no se puede asegurar que el funcionamiento sea el correcto.

1. Obtención de las fuentes.

   - El fichero `postgresql-7.1.3.tar.gz` que contiene los ficheros fuente se encuentra en el CD adjunto. También puede descargarse de la URL:

`ftp.es.postgresql.org/postgresql/source/v7.1.3/postgresql-7.1.3.tar.gz`

2. Descompresión de las fuentes.

   - Situarse en el directorio donde se ha descargado el fichero `postgresql-7.1.3.tar.gz`.
   - Como root, escribir en la línea de comandos:
     `# tar xzvf postgresql-7.1.3.tar.gz -C /usr/src/`

3. Compilación e instalación de los ejecutables.

   - Cambiar al directorio `/usr/src/postgresql-7.1.3`:
     `# cd /usr/src/postgresql-7.1.3`
   - Configurar la instalación para el sistema:
     `# ./configure -with-CXX`

4. Tras la ejecución se creará un fichero 'Makefile' con las opciones de compilación adaptadas a los parámetros de la maquina en la que estamos trabajando, tan sólo queda ejecutar:

   `# make`

   y después:

   `# make install`

5. Postinstalación.

   - Configurar el *linker* `ld` para que enlace las librerías de postgreSQL. Como root:
     - añadir al fichero `/etc/ld.so.conf` la línea:
       `/usr/local/pgsql/lib`
     - ejecutar:
       `# ldconfig`
   - Creación de un usuario para postgreSQL:
     - Teclear, como root:
       `# adduser postgres`
     - Esto añadirá un nuevo usuario de nombre postgres, para dar una contraseña a ese usuario:
       `# passwd postgres`

6. Inicializar postgreSQL.

   - Crear el directorio donde se va a almacenar la base de datos:
     `# mkdir /usr/local/pgsql/data`
   - Hacer que el usuario `postgres` sea el propietario del directorio:
     `# chown postgres /usr/local/pgsql/data`
     `# chgrp postgres /usr/local/pgsql/data`
   - Entrar como el usuario postgres:
     `# su - postgres`

- iniciar la estructura de bases de datos:

  ```
  # /usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
  ```

Con esto se crean las estructura de soporte de las bases de datos que se creen posteriormente.

7. Arrancar postgreSQL. Como usuario postgres, ejecutar:

   ```
   $ /usr/local/pgsql/bin/postmaster -D /usr/local/pgsql/data -i &
   ```

   A partir de este punto postgreSQL se está ejecutando en el sistema. Hay que volver a ejecutar el comando anterior cada vez que se reinicie la máquina.

8. Cambiar los parámetros de postgreSQL.

   Editar el fichero `/usr/local/pgsql/data/postgresql.conf` y sustituir las siguientes entradas:

   - `#fsync = true`

     por

     `fsync = off`

     Con esto se desconecta el modo fsync [27].

   - `#wal_files = 0 # range 0-64`

     por

     `wal_files = 10 # range 0-64`

     Con esto se aumenta el número de ficheros WAL (Write Ahead Logging) [27].

   - `#checkpoint_segments = 3 # in logfile segments (16M each), min 1`

     por

     `checkpoint_segments = 10 # in logfile segments (16M each), min 1`

     Con esto se aumenta el número de ficheros WAL que se deben llenar para que postgreSQL produzca un checkpoint automático [25].

   - `#checkpoint_timeout = 300 # in seconds, range 30-3600`

     por

     `checkpoint_timeout = 3600 # in seconds, range 30-3600`

     Con esto se aumenta el tiempo máximo entre dos checkpoints automáticos [27].

9. Forzar la lectura del fichero de configuración:

   ```
   # killall -HUP postmaster
   ```

   postgreSQL está listo para el test.

En el CD adjunto se incluyen todos los manuales de postgreSQL-7.1.3: manual de administrador, manual de desarrollador, manual programador, manual de referencia, manual de usuario y tutorial.

3.1.2. Instalación de Gnuplot

Gnuplot es una herramienta gráfica que se utiliza en la representación de las gráficas de salida del benchmark. El paquete *rpm*, para máquinas Intel 80X86, necesario para la instalación, es `gnuplot-3.7.1-13.i386.rpm` y se encuentra en el CD adjunto. Además se puede obtener de la página http://www.redhat.com/apps/download.

Tras obtener los paquetes, tecleamos:

```
# rpm -i gnuplot-3.7.1-13.i386.rpm
```

En el CD adjunto se incluye el manual de usuario de Gnuplot.

3.1.3. Instalación del benchmark

En el CD adjunto se incluyen los ficheros fuente en formato *tar.gz* y en *rpm*, y los binarios en formato *rpm* para máquinas intel 80X86. A continuación se detallan los procedimientos de instalación, ya sea instalando el paquete rpm o compilando las fuentes. Es necesario que postgreSQL esté instalado y configurado antes de comenzar el proceso de instalación del benchmark.

3.1.3.1 Instalación del paquete rpm

1. Como root, copiar el archivo `tpcc_uva-1.0-1.i386.rpm` en el directorio `/tmp`.

2. Cambiar al directorio `/tmp`

   ```
   # cd /tmp
   ```

3. Instalar el paquete

   ```
   # rpm -i tpcc_uva-1.0-1.i386.rpm --nodeps
   ```

   Esto creará los directorios `/usr/share/bin` en el que se alojarán los ejecutables del benchmark y `/usr/share/var/tpcc` donde se almacenarán los ficheros de bitácora. Además se instalarán las páginas `man` de cada uno de los programas de que se compone el benchmark en el directorio `/usr/share/man`.

   Nota: El modificador `--nodeps` se utiliza porque el `rpm` no reconoce como instaladas algunas librerías de postgreSQL. Si en la instalación de postgreSQL se han seguido todos los pasos descritos anteriormente, los programas del benchmark van a funcionar correctamente.

4. Modificación de PATH:

Es necesario indicar al sistema operativo la dirección del directorio de los ejecutables.

```
# PATH=$PATH:/usr/share/bin
```

Cada vez que se reinicie el sistema debe repetirse el paso anterior. Otra posibilidad es modificar PATH en el fichero `.bash_profile` del usuario root para evitar este paso.

En este punto el benchmark está preparado para su ejecución.

La desinstalación del benchmark se puede llevar a cabo ejecutando:

```
# rpm -e tpcc_uva-1.0-1.i386.rpm
```

3.1.3.2 Compilación de las fuentes

1. Copiar al directorio `/tmp` el fichero `tpcc_uva-1.0-1.tar.gz` contenido en el cdrom adjunto.

2. Como root, descomprimir las fuentes:

   ```
   # tar xzvf /tmp/tpcc_uva-1.0-1.tar.gz -C /usr/src/
   ```

3. Cambiar al directorio `/usr/src/tpcc_uva-1.0.1`

   ```
   # cd /usr/src/tpcc_uva-1.0.1
   ```

4. Compilar las fuentes:

   ```
   # make
   ```

5. Instalar los ejecutables:

   ```
   # make install
   ```

   Esto creará los directorios `/usr/share/bin` en el que se alojarán los ejecutables del benchmark y `/usr/share/var/tpcc` donde se almacenarán los ficheros de bitácora.

   Además se instalarán las páginas `man` de cada uno de los programas de que se compone el benchmark en el directorio `/usr/share/man`.

6. Modificación de PATH:

   Es necesario indicar al sistema operativo la dirección del directorio de los ejecutables.

   ```
   # PATH=$PATH:/usr/share/bin
   ```

   Cada vez que se reinicie la máquina, se deberá repetir el paso anterior. Otra posibilidad es modificar PATH en el fichero `.bash_profile` del usuario root para evitar este paso.

```
        +-------------------------------------------------+
        | BENCHMARK TPCC  --- UNIVERSIDAD DE VALLADOLID --- |
        +-------------------------------------------------+


        1.- CREAR UNA NUEVA BASE DE DATOS PARA EL TEST.
        2.- RESTAURAR UNA BASE DE DATOS EXISTENTE.
        3.- EJECUTAR EL TEST.
        4.- COMPROBAR LA CONSISTENCIA DE LA BASE DE DATOS.
        5.- ELIMINAR LA BASE DE DATOS.
        6.- VER RESULTADOS DEL TEST.
        7.- COMPROBAR ESTADO DE LA BASE DE DATOS.


        8.- Salir.

SELECCIONE OPERACION:
```

Figura 3.1: Menú de opciones.

En este punto el benchmark está preparado para su ejecución.

La desinstalación del benchmark se puede llevar a cabo ejecutando:

```
# make uninstall
```

3.2. Manejo del benchmark

El benchmark TPCC-UVA debe ejecutarse como usuario root de la siguiente forma:

```
# bench
```

El programa `bench` es el Controlador del Benchmark. Además de proporcionar la interfaz con el usuario, se encarga de activar todos los procesos relacionados en la medición del rendimiento.

La interfaz con el usuario consiste en una pantalla de menú a través de la cual el usuario accede a las funciones disponibles. Esta pantalla se muestra en la figura 3.1

A continuación se describen las distintas opciones a las que pueden acceder el usuario.

3.2.1. Opciones del benchmark

Opción 1: se crea una nueva base de datos poblada según los requisitos del estándar TPC-C. Si no se muestra esta opción, significa que ya existe la base de

datos para el test.

Al seleccionar esta opción el programa solicita al usuario el número de almacenes que contendrá la base de datos y le pide autorización para continuar el proceso. El número de almacenes debe estar comprendido entre 1 y 100, de lo contrario al intoducir el valor se regresará al menú.

El tamaño de la base de datos por cada almacén creado es de 137Mb. La base de datos se crea en el directoio `/usr/local/pgsql/data/base`.

La salida por pantalla es la siguiente:

```
SELECCIONE OPERACION: 1
Introduzca el número de almacenes: 5
¿Continuar? (s/n): s
Comenzando a poblar la base de datos
Hora de comienzo: 2002-20-05 22:50:36
POBLANDO LA BASE DE DATOS PARA 5 ALMACENES...
```

La carga de la base de datos puede detenerse en cualquier momento pulsando simultáneamente las teclas 'control' y 'c' <ctrl-c>. En ese momento el programa pregunta al usuario si desea terminar la carga. En caso afirmativo se detiene el proceso, regresando al menú y en caso negativo se continua en el mismo punto donde se produjo la interrupción.

La salida por pantalla es la siguiente:

```
POBLANDO TABLA ITEM ...

2002-20-05 22:51:49>Creados 10000 Artículos de 100000
2002-20-05 22:53:04>Creados 20000 Artículos de 100000
2002-20-05 22:54:16>Creados 30000 Artículos de 100000


¿Terminar el proceso? (S/N) N
continuando...

2002-20-05 22:55:29>Creados 40000 Artículos de 100000
2002-20-05 22:56:41>Creados 50000 Artículos de 100000
2002-20-05 22:57:57>Creados 60000 Artículos de 100000
...
```

Nótese que la población de la base de datos consiste en inserciones intensivas de datos en disco por lo que el proceso puede durar varias horas dependiendo del número de almacenes seleccionados.

La base de datos se crea con el siguiente número de filas:

Tabla item	100.000 filas.
Tabla `Warehouse`	1 fila por cada almacén seleccionado.
Tabla `District`	10 filas por cada almacén seleccionado.
Tabla `Customer`	30.000 filas por cada almacén seleccionado.
Tabla `History`	30.000 filas por cada almacén seleccionado.
Tabla `Orderr`	30.000 filas por cada almacén seleccionado.
Tabla `Order_Line`	Una media de 300.000 filas por cada almacén seleccionado.
Tabla `New-Order`	9.000 filas por cada almacén seleccionado.
Tabla `Stock`	100.000 filas por cada almacén seleccionado.

NOTA: La tabla `Orderr` se corresponde con la tabla ORDER del estándar TPC-C.

Opción 2: se eliminan los cambios en la base de datos producidos por la ejecución de un test, devolviéndola a su estado previo. Esta opción se muestra en el menú sólo cuando exista una base de datos.

Después de que el usuario seleccione esta opción se le pide confirmación para continuar.

Este proceso es menos intensivo que el proceso de carga por lo que su duración será menor.

Al igual que con la opción 1, el usuario tiene la posibilidad de parar la restauración pulsando simultáneamente las teclas 'control' y 'c' (<ctrl-c>). Tras esto se le pedirá confirmación para parar o no el proceso.

La restauración de la base de datos tiene como desventaja que los resultados de rendimiento son peores que con una base de datos recién creada. A pesar de que tras la restauración se realiza una limpieza en la base de datos, no se logra alcanzar la optimización de los índices como en la creación de la base de datos.

La salida que se muestra por pantalla es la siguiente:

```
SELECCIONE OPERACION: 2
¿Continuar? (s/n): s
Comprobando la base de datos...
        Tabla Warehouse... OK
        Tabla Stock... OK
        Tabla Item... OK
        Tabla District... OK
        Tabla Customer... OK
        Tabla History... OK
        Tabla Order... OK
        Tabla Order-Line... OK
Base de datos OK
   RESTAURANDO LAS TABLAS ALTERADAS POR NEW\_ORDER
        Tabla DISTRICT restaurada.
```

```
Restaurando NEW\_ORDER.
 . .
 . .
```

Como se puede ver, el programa comprueba la base de datos para prevenir que la restauración se haga en una base de datos corrupta, es decir, por ejemplo, en la que falte alguna de las tablas o que los nombres de las tablas no sean los correctos. Si la comprobación es correcta se procede a la restauración de la base de datos, en caso contrario se retorna a la pantalla de menú.

Opción 3: permite la ejecución del test de rendimiento. Esta opción no se muestra si no existe una base de datos.

Al seleccionar esta opción el programa solicita al usuario los parámetros que definen al test, que son:

Número de almacenes. Número de almacenes que se van a considerar en la prueba. Este número ha de ser menor o igual que la cantidad de almacenes que contiene la base de datos.

Número de terminales por almacén En las especificaciones del TPC-C el número de terminales está fijado en diez por almacén, sin embargo podemos elegir realizar una prueba con una carga de terminales menor. Sobre esta opción se hablará en la sección 3.2.1.3.

Periodo de rampa. Al inicio del periodo de rampa se lanzan los terminales. El motivo de este periodo es que la tasa de rendimiento se estabilice para comenzar el periodo de medida en estado estable. La duración de este periodo se introducirá en minutos. El rendimiento del sistema durante el periodo de rampa no se considerará en el cálculo final. Un valor adecuado para este periodo son 20 m, aunque el valor optimo depende del sistema y debe comprobarse a partir de la gráfica de evolución de rendimiento (ver apartado 3.2.1.3).

Periodo de medida. En esta entrada se especifica el tiempo en el cual se va a calcular el rendimiento. El estándar TPC-C especifica que este periodo ha de tener una duración mínima de 2 horas y máxima de 8. La duración de este periodo se introducirá en minutos.

Tras introducir los datos anteriores se pide confirmación para continuar. En este punto el usuario puede abortar el test si detecta que alguno de los valores anteriores es incorrecto. La salida por pantalla es la siguiente:

```
SELECCIONE OPERACION: 3
->Introduzca el número de almacenes 1
->Introduzca el número de terminales por almacén (máx 10) 10
->Introduzca el intervalo de rampa (minutos) 20
```

```
->Introduzca el intervalo de medida (minutos) 480
¿Continuar? (s/n): s
```

Si los datos anteriores son incorrectos, se informará de tal circunstancia y se volverá a solicitar la introducción de los datos. Para ser considerados como válidos, el número de almacenes deberá estar comprendido entre 1 y 100, el número de terminales entre 1 y 10, y los periodos de rampa y de medida deberán ser mayores que cero. La salida por pantalla es la siguiente:

```
SELECCIONE OPERACION: 3
->Introduzca el número de almacenes 0
->Introduzca el número de terminales por almacén (máx 10) 0
->Introduzca el intervalo de rampa (minutos) 0
->Introduzca el intervalo de medida (minutos) 0
Error:->Datos incorrectos
->Introduzca el número de almacenes
```

Limpiezas de la base de datos. Si los datos introducidos son correctos y se decide continuar con el test, se pregunta al usuario si desea realizar limpiezas (*vacuums*) en la base de datos durante el test.

A medida que postgreSQL realiza transacciones, las eliminaciones y las modificaciones de las filas de la base de datos van dejando información residual en las tablas. La acumulación de esta información reduce en gran medida el rendimiento, ya que relentiza la ejecución de las transacciones.

Si el usuario no desea realizar limpiezas se continuará con la ejecución del test. De lo contrario, se le solicitará el intervalo entre ellas. Durante el test las limpiezas se realizarán periódicamente, con una separación entre ellas igual al intervalo seleccionado. La primera de ellas se realizará una vez transcurrido dicho intervalo tras el inicio del test. La salida por pantalla es la siguiente:

```
->¿Desea realizar limpiezas (VACUUM) en la base de datos? (s/n):s
->Introduzca el intervalo entre vacuums (minutos): 60
```

Para ser considerado como válido el intervalo entre limpiezas ha de ser mayor que cero. Si se introduce un intervalo incorrecto se informa de tal circunstancia, y se volverá a solicitar. La salida por pantalla es la siguiente:

```
->Introduzca el intervalo entre vacuums (minutos): 0
->Intervalo no válido.
->Introduzca el intervalo entre vacuums (minutos):
```

El valor óptimo del intervalo depende de las características del sistema, por lo que debe ser ajustado tras sucesivas pruebas. Se recomienda comenzar con un intervalo de 60 minutos, para después reducir o aumentar el valor dependiendo del rizado de la gráfica de evolución de rendimiento (ver apartado 3.2.1.3).

Una vez introducido el intervalo, se pregunta al usuario si desea fijar el número máximo de limpiezas. Esto permitirá planificarlas, impidiendo, por ejemplo, que se realice una limpieza poco antes de que finalice el test, ya que en este caso no se aprovecharía el aumento en la velocidad de ejecución las transacciones. Si no se desea fijar el número, las limpiezas se realizan hasta que finalice el test. La salida por pantalla es la siguiente:

```
->¿Desea fijar el número máximo de vacuums? (s/n): s
->Introduzca el número máximo de vacuums: 4
```

Si se introduce un número máximo menor que cero, se considerará como inválido. En este caso se informa al usuario de tal circunstancia y se volverá a solicitar el dato. La salida por pantalla es la siguiente:

```
->¿Desea fijar el número máximo de vacuums? (s/n): s
->Introduzca el número máximo de vacuums: -3

->Opción no válida.
->Introduzca el número máximo de vacuums:
```

Si como número máximo se introduce cero, las limpiezas también se realizarán hasta que finalice el test.

Las operaciones de limpieza sólo son recomendables en periodos de test superiores a tres horas, ya que en periodos menores la disminución del rendimiento no es tan pronunciada.

Una vez introducidos los datos para las limpiezas se muestran las opciones seleccionadas y se pide confirmación para continuar:

```
->Ha elegido realizar vacuums cada 60 minutos, con un número máximo de 4.
¿Continuar? (s/n):
```

Si se desea continuar, el programa pasa a comprobar si la base de datos contiene las filas suficientes para ejecutar el test, con el número de almacenes especificado. Este número de filas es el que se especifica en la cláusula 4 del estándar TPC-C como población mínima necesaria para la realización de un test. La salida por pantalla es la siguiente:

```
¿Continuar? (s/n): s
Comprobando la base de datos...
        Tabla Warehouse... OK
        Tabla Stock... OK
        Tabla Item... OK
        Tabla District... OK
        Tabla Customer... OK
        Tabla History... OK
        Tabla Order... OK
        Tabla Order-Line... OK
        Tabla New-Order... OK
Base de datos OK
```

Si la comprobación para alguna de las tablas es errónea, se pide al usuario confirmación para continuar, en caso contrario se continúa con el test. Si el error se produce en la tabla `New-Order` (no aparece `OK` después del nombre de la tabla), es posible realizar el test sin riesgo de que el programa falle, ya que las transacciones que operan sobre la tabla New-Order contemplan la posibilidad de que la tabla tenga un número de filas menor al de la población inicial. El buen estado del resto de tablas es imprescindible para el correcto funcionamiento del programa. Si se continúa, se producirán errores en la ejecución de las transacciones. Si, por ejemplo, el usuario ha introducido un número de almacenes para realizar el test mayor que el existente en la base de datos, la comprobación de la tabla `Warehouse` será errónea. En este caso se recomienda no continuar con el test.

En caso de comprobación errónea la salida por pantalla es la siguiente:

```
¿Continuar? (s/n):s
Comprobando la base de datos...
Tabla Warehouse...
Error: La base de datos no está poblada correctamente
Continuar (s/n)
```

Si tras la comprobación de la base de datos se desea continuar se lanza el Monitor de Transacciones (MT), y los Emuladores de Terminal Remoto (ETR). La salida por pantalla es la siguiente:

```
Lanzando tm... OK
Lanzando terminales... OK
Lanzando controlador de vacuums... OK
```

Una vez lanzados, el MT escribirá por pantalla información a cerca de los mensajes que va ejecutando: en primer lugar los mensajes de conexión de los ETR con el MT, y después los de transacción.

La salida por pantalla es la siguiente:

```
>> CONEXION. SEM 1277959. SHM 350453779. Asignado Terminal 4.
       ..
       ..
       ..

>> TRANSACCION NEW_ORDER.    Terminal 4.        EJECUTADA.
>> TRANSACCION PAYMENT.      Terminal 9.        EJECUTADA.
       ..
       ..
       ..
```

Este proceso continuará hasta que finalicen los periodos de rampa y de medida seleccionados por el usuario. En cualquier momento el usuario puede terminar el test pulsando simultáneamente las teclas 'control' y 'c' (<ctrl-c>). Si lo hace se le solicitará confirmación. Debido a que el test no se puede parar es probable que el usuario no vea en la pantalla el mensaje de solicitud de confirmación para finalizar el test, ya que se seguirán escribiendo las transacciones ejecutadas. El usuario deberá introducir `'s'` o `'n'` dependiendo de si desea o no finalizarlo. En caso afirmativo, no finalizara inmediatamente, si no que habrá que esperar a la desactivación de todos lo procesos involucrados.

Si se ha parado el test mediante <ctrl-c>, cuando se ejecute un recuento de resultados los datos se ajustarán al tiempo transcurrido. Si se finaliza el test antes de que comience el periodo de medida, no se podrá llevar a cabo un recuento de resultados. El recuento de resultados se explicará posteriormente.

Opción 4: Se comprueba que la base de datos cumple las cuatro condiciones de consistencia especificadas en las cláusulas 3.3.2.1, 3.3.2.2, 3.3.2.3, 3.3.2.4, estándar TPC-C. Nótese que, aunque en el estándar se definen doce condiciones de consistencia, sólo se requiere la demostración explícita de las cuatro primeras. Esta operación permite al usuario comprobar que la base de datos se encuentra en un estado consistente tras su creación, tras su restauración o tras la ejecución de un test de rendimiento.

La opción de comprobación de consistencia sólo se muestra en caso de que exista una base de datos.

Al seleccionar esta opción, el programa pide autorización para continuar. En caso afirmativo, se comprobará la consistencia de la base de datos. Por cada condición se mostrará un mensaje informando del resultado de la comprobación. La salida por pantalla es la siguiente:

```
SELECCIONE OPERACION: 4
¿Continuar? (s/n)s
Numero de almacenes 1
```

```
EL ALMACÉN 1 CUMPLE LA CONDICION 1
 SE HA CUMPLIDO LA CONDICIÓN 2 EN TODOS LOS DISTRITOS
 SE HA CUMPLIDO LA CONDICIÓN 3 EN TODOS LOS DISTRITOS
 SE HA CUMPLIDO LA CONDICIÓN 4 EN TODOS LOS DISTRITOS
```

Si no se ha cumplido alguna de las condiciones se mostrará un mensaje como el siguiente:

```
SELECCIONE OPERACION: 4
¿Continuar? (s/n)s
Numero de almacenes 1
EL ALMACÉN 1 NO CUMPLE LA CONDICION 1
```

Opción 5: Se elimina la base de datos existente.

Al seleccionar esta opción, el programa solicita confirmación para eliminar la base de datos. Si el resultado es afirmativo, se elimina la base de datos de test existente. En caso negativo se regresa al menú.

```
SELECCIONE OPERACION: 5
¿Está seguro de que quiere eliminar la base de datos?: s/n s

La base de datos ha sido eliminada.

Continuar ...
```

Opción 6: Se realiza el análisis de las bitácoras de rendimiento para la obtención del rendimiento ofrecido por el sistema.

Al seleccionar esta opción el programa examina los ficheros de datos del test, muestra por pantalla la información resultante, imprime los ficheros de información de los checkpoints y de las limpiezas efectuadas durante el test y genera los ficheros de datos necesarios para generar las gráficas definidas en las cláusulas 5.6.1, 5.6.3 y 5.6.4 del estándar TPC-C. Los ficheros de salida se almacenan en el directorio actual.

A continuación se incluye una salida de esta opción:

```
Recuento del test realizado el 2002-25-05 a las 12:49:53 para 4 almacenes.

Inicio del periodo de medida: 10.001267 m

Fin del periodo de medida: 490.002100 m
```

```
RENDIMIENTO OBTENIDO: 43.617 tpmC para 4 almacenes.

48135 Transacciones en total.

TRANSACCIONES NEW ORDER:
  20936 Transacciones en periodo de medida. 21410 Totales.
  Porcentaje: 43.494%
  Porcentaje bien hechas : 94.292%
  Tiempo de respuesta: min/med/max/90th: 0.033/3.640/262.981/3.720
  Porcentaje de transacciones rechazadas: 1.008% .
  Número medio de artículos por orden: 9.924 .
  Porcentaje de artículos remotos: 1.003% .
  Tiempo de pensar min/med/max: 0.000/12.043/115.000
Continuar...

TRANSACCIONES PAYMENT:
  20931 Transacciones en periodo de medida. 21403 Totales.
  Porcentaje: 43.484%
  Porcentaje bien hechas : 94.023%
  Tiempo de respuesta: min/med/max/90th: 0.007/3.197/262.317/3.720
  Porcentaje de transacciones remotas: 15.303% .
  Porcentaje de clientes seleccionados por C_ID: 40.256% .
  Tiempo de pensar min/med/max: 0.000/11.995/120.000
  Continuar...

TRANSACCIONES ORDER STATUS:
  2089 Transacciones en periodo de medida. 2138 Totales.
  Porcentaje: 4.340%
  Porcentaje bien hechas : 93.394%
  Tiempo de respuesta: min/med/max/90th: 0.026/3.612/243.927/4.080
  Porcentaje de clientes seleccionados por C_ID: 43.226% .
  Tiempo de pensar min/med/max: 0.000/9.875/90.000
Continuar...

TRANSACCIONES DELIVERY:
  2087 Transacciones en periodo de medida. 2137 Totales.
  Porcentaje: 4.336%
  Porcentaje bien hechas : 100.000%
  Tiempo de respuesta: min/med/max/90th: 0.000/0.000/0.193/0.000
  Porcentaje ejecucion < 80s : 99.138%
  Tiempo de ejecucion min/med/max: 0.537/4.469/242.659
  N° de distritos saltados: 0 .
  Porcentaje de distritos saltados: 0.000%.
  Tiempo de pensar min/med/max: 0.000/4.978/45.000
  Continuar...

TRANSACCIONES STOCK LEVEL:
```

```
  2092 Transacciones en periodo de medida. 2137 Totales.
  Porcentaje: 4.346%
  Porcentaje bien hechas : 97.610%
  Tiempo de respuesta: min/med/max/90th: 0.022/4.441/237.119/4.640
  Tiempo de pensar min/med/max: 0.000/4.851/45.000
Continuar...

Checkpoints con mayor duración:
Hora de comienzo        Tiempo desde el Inicio (s)  Duración (s)
Sat May 25 15:00:08 2002    7814.349000        22.720000
Sat May 25 19:31:05 2002    24071.936000        11.895000
Sat May 25 15:30:30 2002    9637.112000        5.301000
Sat May 25 20:31:20 2002    27686.947000        5.210000
Continuar...

Vacuums con mayor duración:
Hora de comienzo Tiempo desde el Inicio (s) Duración (s)
Sat May 25 18:18:16 2002 19702.855000 451.607000
Sat May 25 19:25:48 2002 23754.511000 450.266000
Sat May 25 16:04:00 2002 11646.438000 429.448000
Sat May 25 17:11:09 2002 15675.919000 426.883000

>> TEST PASADO
Continuar...
```

Los datos referentes a las limpiezas (*Vacuums*) de mayor duración sólo se muestran en caso de que se hayan realizado limpiezas en la base de datos durante el test.

Al final de todos los datos aparece un mensaje indicando la validez el test.

Después de mostrar los resultados por pantalla, se pregunta al usuario si desea almacenar esos mismos resultados en un fichero. En ese caso se solicita el nombre del fichero donde desea guardarlos y se escribe en el directorio actual. La salida por pantalla es la siguiente

```
¿Desea guardar los resultados en un fichero? (s/n): s
Introduzca el nombre del fichero (máximo 15 caracteres)
->t_25_5.txt
Los resultados han sido escritos en el fichero t_25_5.txt .
Continuar...
```

En el apartado 3.2.1.3 explicará el significado y el análisis de los resultados de rendimiento.

Opción 7: Se comprueba el número de filas de cada tabla en la base de datos.

En esta opción de se hace un recuento del número de filas que contiene cada una de las tablas de la base de datos, para que el usuario pueda ver en qué estado se encuentra.

La salida que se muestra por pantalla tras seleccionar esta opción es:

```
 TAMAÑO DE LAS TABLAS:

->Tabla WAREHOUSE: 1 fila.
->Tabla DISTRICT: 10 filas.
->Tabla ITEM: 100000 filas.
->Tabla STOCK: 100000 filas.
->Tabla CUSTOMER: 30000 filas.
->Tabla HISTORY: 30000 filas.
->Tabla ORDERR: 30000 filas.
->Tabla ORDER-LINE: 300282 filas.
->Tabla NEW-ORDER: 9000 filas.


Continuar...
```

De esta forma el usuario podrá saber si se ha hecho un test anterior, y podrá decidir si crear de nuevo una base de datos, restaurarla o seguir utilizándola. Además se puede averiguar el número de almacenes que contiene la base de datos.

Esta opción puede ser útil para prevenir el hacer un test sobre una base de datos mal dimensionada. Se debe tener en cuenta que las especificaciones del TPC-C, indican que el test de rendimiento ha de realizarse con las tablas pobladas con el número de filas que se ha indicado en la explicación de la opción 1.

Opción 8: Permite al usuario salir del programa.

3.2.2. Obtención de resultados

En este apartado se describirá el proceso para la obtención de los resultados de rendimiento del sistema usando el benchmark TPCC-UVA.

3.2.1.1 Creación de una base de datos para el test

El primer paso para la obtención de los resultados de rendimiento, es crear una base de datos para la realización del test. Para ello se selecciona la opción 1 del

menú del benchmark y se introduce el número de almacenes con que se quiere poblar la base de datos.

3.2.1.2 Realización del test de rendimiento

Una vez finalizado el proceso anterior, se procede a la ejecución del test. Para ello se selecciona la opción 3 del menú del benchmark. Se tendrán que introducir los parámetros con que se desea que se ejecute dicho test, que son:

- Número de almacenes con que se desea realizar el test que deberá ser menor o igual que el número con que se pobló la base de datos.
- Número de terminales con que se desea realizar el test. Para la ejecución del test deberán ser 10 terminales.
- Intervalo de rampa: Periodo de tiempo que se desea que se espere antes de realizar la medición de rendimiento, para que el sistema alcance un estado estable, según se especifica en la cláusula 5.5 del estándar TPC-C.
- Intervalo de medida: Periodo de tiempo en el que se efectuará la medición de rendimiento.
- Si se desea realizar limpiezas periódicas en la base de datos durante el test, se especificará el intervalo entre ellas. La primera de ellas se realizará una vez transcurrido el intervalo seleccionado. Si también se desea limitar el número máximo de limpiezas, se especificará dicho número.

Una vez seleccionados los parámetros anteriores comenzará el test, y deberá esperarse a su finalización.

3.2.1.3 Análisis de resultados

El rendimiento del sistema está determinado por el número de transacciones de tipo New-Order por minuto (tpmC) ejecutadas durante el periodo de medida. Las transacciones deben haberse realizado bajo las condiciones definidas en el estándar TPC-C. En los apartados siguientes, se detallan los pasos para obtener la información anterior, así cómo la información adicional requerida por el estándar para considerar los resultados válidos.

Resultados del Test

Finalizado el test, se pueden ver los resultados seleccionando la opción 6 del menú del benchmark. El programa, tras procesar los ficheros de bitácora de los

Emuladores de Terminal Remoto y de Monitor de Transacciones, muestra la información acerca del rendimiento y del test ejecutado. De los datos que proporciona el programa, dos son los que verdaderamente determinan el rendimiento de la máquina: el número de transacciones por minuto tpmC, y los tiempos de respuesta de transacción. El resto determinan si la carga de trabajo del test se ha realizado según las especificaciones del TPC-C. El formato de los datos se salida se muestra en la explicación de la opción 6 del benchmark. Estos datos son:

- Fecha en la que se realizó el test.
- Instante de comienzo y de finalización del periodo de medida en minutos respecto del inicio del test.
- Rendimiento obtenido en transacciones por minuto (tpmC) y número de almacenes con que se ejecutó el test. Esta es la cantidad de transacciones New-Order realizadas por minuto y no el número total de transacciones por minuto.
- Número total de transacciones ejecutadas.

Para cada tipo de transacción se muestra:

- Número de transacciones de ese tipo ejecutadas durante el periodo de medida, así como el número total.
- Porcentaje de transacciones de ese tipo con respecto al total. Para que sea considerado como válido, las transacciones Payment deben superar el 43 % y las transacciones Order-Status, Delivery y Stock-Level el 4 % (ver cláusula 5.2.3 del estándar TPC-C).
- Porcentaje de transacciones bien hechas. Representa el porcentaje de transacciones cuyo tiempo de respuesta de transacción es menor que el especificado por el estándar TPC-C. Estos tiempos son:
  - Para las transacciones New-order, Payment, Order-Status y Delívery, 5 segundos.

    Nota: En el caso de la transacción Delivery este tiempo se refiere al tiempo de encolado, no de ejecución (ver perfil de transacción Delivery de la cláusula 2.7 estándar TPC-C).
  - Para la transacción Order-Status 20 segundos.

  Nota: El tiempo de respuesta de transacción se define como el tiempo transcurrido desde que el terminal envía la transacción hasta que recibe los resultados. (ver definición de tiempo de respuesta de transacción en la cláusula 5.3.4 del estándar TPC-C).

Para que el rendimiento obtenido sea válido según las especificaciones del estándar TPC-C, el 90 % de las transacciones han de tener un tiempo de respuesta de transacción inferior a los anteriores. En el caso de que este tiempo sea mayor que cuatro veces el máximo especificado por el estándar se mostrará el simbolo (><) que indica que no se ha podido calcular.

- Tiempo de Respuesta de Transacción mínimo, medio, máximo, y del 90 % de las transacciones, es decir tiempo por debajo del cual se ejecutan el 90 % de las transacciones.

  Para que el rendimiento obtenido sea válido según las especificaciones del estándar TPC-C, el tiempo de respuesta del 90 % de transacción ha de ser menor que los anteriormente citados.

- Tiempos de pensar mínimo, medio, y máximo. Estos tiempos se simulan en los ETR para imitar el tiempo en el que el usuario toma decisiones. Para cada tipo de transacción, el estándar TPC-C especifica que los tiempos medios mínimos de pensar de los terminales han de ser:

  - Para las transacciones New-Order y Payment, 12 segundos.
  - Para la transacción Order-Status, 10 segundos.
  - Para las transacciones Delivery y Stock-Level, 5 segundos.

Además para la transacción New-Order se muestra:

- Porcentaje de transacciones canceladas (rollback). El estándar TPC-C especifica que este porcentaje ha de ser del 1 %, permitiéndose una variación del $\pm 0,1$ %.
- Número medio de artículos por cada orden. El estándar TPC-C especifica que este número medio ha de ser 10, permitiéndose una variación de $\pm 0,5$.
- Porcentaje de artículos remotos. El estándar TPC-C especifica que este porcentaje ha de ser del 1 %, permitiéndose una variación de $\pm 0,05$ %. Si el test se ha ejecutado con un solo almacén no se mostrará este dato.

Para la transacción Payment se muestra:

- Porcentaje de transacciones Payment remotas. El estándar TPC-C especifica que este porcentaje ha de ser del 15 %, permitiéndose una variación de ± 1 %. Si el test se ha ejecutado con un solo almacén no se mostrará este dato.
- Porcentaje de clientes seleccionados por C_ID (identificador de clientes). El estándar TPC-C especifica que este porcentaje ha de ser del 40 %, permitiéndose una variación de ± 3 %.

Para la transacción Order-Status se muestra:

- Porcentaje de clientes seleccionados por C_ID (identificador de clientes). El estándar TPC-C especifica que este porcentaje ha de ser del 40 %, permitiéndose una variación de ±3 %.

Para la transacción Delivery se muestra:

- Porcentaje de transacciones con un tiempo de ejecución menor de 80 segundos. El estándar TPC-C especifica que este porcentaje ha de ser por lo menor del 90 %.

  Nota: El tiempo de ejecución para la transacción Delivery se define como el tiempo transcurrido desde que el terminal envía la transacción hasta que el Monitor de Transacciones finaliza su ejecución. (ver definición de tiempo de ejecución en la cláusula 2.7.2.2 del estándar TPC-C).

- Tiempo de ejecución mínimo, medio, máximo, y del 90 % de las transacciones (es decir tiempo por debajo del cual se ejecutan el 90 % de las transacciones). El estándar especifica que el tiempo de ejecución del 90 % de transacciones Delivery ha de ser menor de 80 segundos.

- Número y porcentaje de distritos saltados como resultado de no encontrarse ninguna orden pendiente de reparto. Este dato sirve para informar si se ha saltado algún distrito en más del uno por ciento de los distritos procesados o ha ocurrido en más de una ocasión, como se especifica en la cláusula 2.7.4.2 del estándar.

Además de lo anterior se muestra:

- Checkpoints de mayor duración:

  Se muestra información acerca de los cuatro *checkpoints* de mayor duración ordenados por orden descendente. Este benchmark ejecuta checkpoints cada 30 minutos comenzando en el inicio del periodo de medida (en la cláusula 5.5.2.2 del estándar TPC-C se describe el termino *checkpoint* y las consideraciones acerca de su ejecución durante el test de rendimiento).

  La información que se muestra es:

  - Fecha y hora de comienzo del checkpoint.
  - Tiempo de test transcurrido al comienzo del checkpoint, expresado en segundos.
  - Duración en segundos del checkpoint.

- Limpiezas de mayor duración:

  Se muestra información acerca de los cuatro limpiezas de mayor duración ordenados por orden descendiente.

  La información que se muestra es:

  - Fecha y hora de comienzo de la limpieza.
  - Tiempo de test transcurrido al comienzo de la limpieza, expresado en segundos.
  - Duración en segundos de la limpieza.

Tras mostrar la anterior información, el benchmark informa si se han cumplido los requisitos en cuanto al tiempo de respuesta. En caso afirmativo se muestra el mensaje:

```
>> TEST PASADO
```

En caso contrario se muestra:

```
>> TEST FALLIDO
No se han cumplido los requisitos de tiempo de respuesta.
Consulte la sección 'Análisis de Resultados' del manual de usuario.
```

Si el test resulta *FALLIDO* es porque el sistema no soporta la carga de trabajo indicada. Será necesario, entonces, realizar una nueva prueba con un número menor de almacenes.

Se recomienda almacenar la información anterior en un fichero como sugiere el programa.

Archivos de información acerca del test

Durante el proceso de cálculo, del paso anterior, se genera una serie de ficheros que contienen información a cerca del cómo se ha llevado a cabo el test. Dichos ficheros son:

Fichero `'medida.log'`. Este fichero se encuentra en el directorio `/usr/share/var/tpcc` después de realizar el test. Proporciona la siguiente información.

- Fecha y hora en la que se realizó el test.
- Número de almacenes con que se realizó el test.

- Número de terminales con que se realizó el test.
- Intervalo de medida en minutos.
- Intervalo de rampa en minutos.
- Intervalo entre limpiezas, en caso de haberse realizado. En caso contrario este valor será cero.
- Número máximo entre limpiezas, si se ha configurado. De no haberlo hecho este valor será cero, indicando que se han realizado hasta la finalización del test.
- Sello de hora del comienzo del test, con precisión de milisegundos.
- Sello de hora del comienzo del intervalo de medida, con precisión de milisegundos.
- Sello de hora del de finalización del intervalo de medida, con precisión de milisegundos.
- Sello de hora del de finalización del intervalo de medida, con precisión de milisegundos.
- Sello de hora de finalización del test, con precisión de milisegundos.

Fichero `'check.dat.'`**.** Se encuentra en el directorio donde se llamó al programa `'bench'`, y se ejecutó un recuento mediante la opción 6. Proporciona información sobre todos los checkpoints realizados en la base de datos durante el último test. Esta información es la siguiente:

- Fecha y hora del inicio de cada checkpoint.
- Tiempo de test transcurrido al comienzo del 'checkpoint', expresado en segundos.
- Duración en segundos del 'checkpoint'.

Fichero `'vacuum.dat'`**.** Se encuentra en el directorio donde se llamó al programa `'bench'`, y se ejecutó un recuento mediante la opción 6. Proporciona información sobre todos las limpiezas que se realizaron en la base de datos durante el último test, en caso de haber elegido esta opción. Esta información es la siguiente:

- Fecha y hora del inicio de cada limpieza.
- Tiempo de test transcurrido al comienzo del la limpieza, expresado en segundos.
- Duración en segundos del la limpieza.

Fichero `'tm_err.dat'`**.** Se encuentra en el directorio `/usr/share/var/tpcc/`. Lo genera el Monitor de Transacciones durante la ejecución del test. Contiene los posibles mensajes de error resultantes de la ejecución de transacciones.

Fichero de datos para las gráficas. Estos ficheros contienen los datos necesarios para la visualización de los gráficos especificados en las cláusulas 5.6.1, 5.6.3 y 5.6.4. Se encuentran en el directorio donde se llamó al programa `'bench'`, y se ejecutó un recuento mediante la opción 6.

- Ficheros `g1New_Order.dat`, `g1Payment.dat`, `g1OrderStatus.dat`, `g1Delivery.dat` y `g1StockLevel.dat`. Contienen los puntos para la visualización de la gráfica especificada en la cláusula 5.6.1 del TPC-C, para cada uno de los cinco tipos de transacciones ejecutadas durante el test.
- Fichero `g3.dat`. Contiene los puntos para la visualización de la gráfica de la cláusula 5.6.3.
- `g4.dat`. Contiene los puntos para la visualización de la gráfica de la cláusula 5.6.4.

A continuación se explicará el significado y el modo de visualizar estas gráficas.

Obtención y visualización de la gráficas

En este apartado se especifican los pasos necesarios para la obtención de los gráficas que se especificadas en la cláusula 5.6 de las especificaciones del TPC-C.

Gráfica 1. Esta gráfica se especifica en la cláusula 5.6.1 del estándar TPC-C. Representa la distribución frecuencial de los tiempos de respuesta de cada tipo de transacción. El eje 'x' representa el tiempo de respuesta de transacción expresado en segundos. El eje 'y' representa el número de transacciones completadas en ese tiempo.

Para visualizarla se utiliza el programa Gnuplot. El usuario deberá abrir un terminal gráfico, y situarse en el directorio donde se realizó el recuento, que será donde se encuentren los ficheros de datos necesarios. Tras esto invocará a Gnuplot mediante la siguiente sentencia:

```
# gnuplot
```

Tras lo anterior aparece la pantalla de Gnuplot donde se muestra un prompt. Para visualizar cada una de las gráficas se deberá introducir la siguiente sentencia en dicho prompt:

```
gnuplot>plot 'g1New_Order.dat' with lines.
```

Una vez visualizada la gráfica se podrán añadir elementos los elementos distintivos:

- Título de la gráfica: se puede titular la gráfica por medio de las sentencias:

```
>set title 'Distribución del Tiempo de Respuesta'
>replot
```

- Nombre de los ejes: para añadir los nombres de los ejes se utilizarán las siguientes sentencias:

  ```
  >set xlabel 'Tiempo de Respuesta (s)'
  >set ylabel 'Numero de Transacciones'
  >replot
  ```

- Línea de Tiempo Medio de Respuesta: Si *tm* es el tiempo medio de respuesta de la transacción correspondiente, se obtendrá una línea por medio de la sentencia:

  ```
  >set arrow 1 from tm, 0 to tm, fin
  >replot
  ```
  donde *fin* es la coordenada y del fin de línea.

- Línea del Tiempo para el 90 %. Si *t90* es el tiempo de respuesta para el 90 % de la transacciones. Se obtiene mediante:

  ```
  >set arrow 2 from t90, 0 to t90, fin
  >replot
  ```

  donde *fin* es la coordenada y del fin de línea.

- Si se quiere añadir una etiqueta en un punto determinado (x,y), se utilizarán las siguientes sentencias:

  ```
  >set label n 'texto' at x, y
  >replot
  ```

  donde *texto* es el texto que contendrá la etiqueta y n un número que identifica a la etiqueta.

Para eliminar cada uno de los elementos que se han dibujado:

- título:

  ```
  >set title ''
  >replot
  ```

- etiquetas de los ejes:

  ```
  >set xlabel ''
  >set ylabel ''
  >replot
  ```

- líneas:
  - Una línea:

    ```
    >set noarrow n
    >replot
    ```

    donde n es el identificador de la línea.
  - Todas las líneas:

    ```
    >set noarrow
    >replot
    ```

- etiquetas:
  - Una etiqueta:

    ```
    >set nolabel n
    >replot
    ```

    donde n es el identificador de la línea.
  - Todas las etiquetas:

    ```
    >set nolabel
    >replot
    ```

Las anteriores sentencias permiten visualizar el gráfico 1 para la transacción New-Order. Análogamente se pueden visualizar el gráfico 1 del resto de transacciones utilizando los ficheros correspondientes: `g1Payment.dat`, `g1OrderStatus.dat`, `g1StockLevel.dat` y `g1Delivery.dat`.

Gráfica 2. Este gráfico se especifica en la cláusula 5.6.2 del estándar TPC-C. Representa variación del tiempo de respuesta de la transacción New-Order en función del rendimiento obtenido. En el eje 'x' se representa la tasa de rendimiento relativa al máximo rendimiento ofrecido por el sistema durante el test. En el eje 'y' se representa el tiempo de respuesta por debajo del cual se ejecutan el 90 % de las transacciones New-Order.

La obtención de los puntos de este gráfico corre a cargo del usuario, que deberá realizar distintos tests modificando la carga de trabajo.

El estándar TPC-C especifica que se deberán obtener al menos tres puntos para este gráfico, aproximadamente al 50 %, 80 % y 100 % del rendimiento máximo ofrecido por el sistema.

El punto para el 100 % corresponde al rendimiento máximo obtenido para el sistema.

Cada punto adicional se obtiene realizando los siguientes pasos. Se recomienda guardar los ficheros de resultados obtenidos el recuento del test antes de proceder a la obtención de los puntos, para evitar sobreescribirlos al realizar dichos pasos.

- Realizar un test con los siguientes parámetros:

  Número de almacenes: El mismo utilizado para obtener el rendimiento máximo.

  Número de terminales: Se utiliza este parámetro para variar la carga de trabajo. Así pues, para obtener el punto aproximado para el 10 % del rendimiento máximo, se utilizará un terminal por almacén. Para el punto del 20 %, dos terminales... y así sucesivamente.

  Periodo de rampa: El mismo utilizado para obtener el rendimiento máximo.

  Periodo de medida: Mínimo 20 minutos.

  Limpiezas: Deben configurarse como en el test en el que se obtuvo el rendimiento máximo.

  Nota: Cuando se realicen sucesivos test es probable que la comprobación de la tabla New-Oreder sea incorrecta, debido a que en tests anteriores se han eliminado filas en dicha tabla. El usuario puede continuar el test sin riesgo de que se produzcan errores en la ejecución.

- Realizar el recuento de resultados y anotar el rendimiento en tpmC obtenido, y el tiempo por debajo del cual se ejecutan el 90 % de las transacciones. Este representa el punto del gráfico buscado.

Una vez obtenidos todos los puntos, se escribirán un fichero, ordenados en orden creciente de terminales utilizados, con el siguiente formato:

```
x1 y1
. .
xi yi
. .
xn yn
```

donde *xi* representa la tasa de rendimiento relativo obtenido expresada en tanto por ciento e *yi* representa el tiempo de respuesta del 90 % de las transacciones New-Order obtenido.

La gráfica determinada por los puntos anteriores se puede visualizar llamando al programa Gnuplot como se hizo para el gráfico 1. El fichero que ha de utilizarse en este caso es el mencionado anteriormente.

```
gnuplot>plot 'fichero.dat' with lines.
```

Una vez visualizada la gráfica se podrán añadir los elementos los elementos distitivos:

- Título de la gráfica: se puede titular la gráfica por medio de las sentencias:

  ```
  >set title 'Variación del Rendimiento'
  >replot
  ```

- Nombre de los ejes: para añadir los nombres de los ejes se utilizarán las siguientes sentencias:

  ```
  >set xlabel 'Tasa de Rendimiento(%)'
  >set ylabel ' Tr del 90% '
  >replot
  ```

- Para obtener la líneas verticales en dos distintos puntos de rendimiento, se utilizarán las siguientes sentencias:

  ```
  >set arrow 'n' from Porcen, 0 to Porcen, fin
  >replot
  ```

  donde *Porcen* es el rendimiento en % del punto correspondiente, *fin* es la coordenada y del fin de línea, y n es un número natural que identifica a la línea.

- Para añadir una línea horizontal con el tiempo de respuesta para el 90 % de las transacciones, se utilizarán las siguientes sentencias:

  ```
  >set arrow 'n' from 0, Tr to 0, fin
  >replot
  ```

  donde *Tr* es el tiempo de respuesta para el 90 % de las transacciones, *fin* es la coordenada x del fin de línea, y n es el número de la línea.

Si se quieren eliminar los elementos dibujados, se deben seguir los pasos descritos en la gráfica 1.

Gráfica 3. Esta gráfica se especifica en la cláusula 5.6.3 del estándar TPC-C. Representa la distribución frecuencial de los tiempos de pensar para la transacción

New-Order. El eje 'x' representa el tiempo de pensar en segundos. El eje 'y' representa el número de transacciones con ese tiempo de pensar.

El fichero que ha de utilizarse en este caso es `g3.dat`, y se encuentra en el directorio donde se llamó al programa y se ejecutó el recuento. Para visualizar la gráfica se utilizará la siguiente sentencia, en el programa Gnuplot:

```
gnuplot>plot 'g3.dat' with lines.
```

Una vez visualizada la gráfica se podrán añadir los elementos los elementos distintivos:

- Título de la gráfica: se puede titular la gráfica por medio de las sentencias:

  ```
  >set title 'Distribución del Tiempo de Pensar'
  >replot
  ```

- Nombre de los ejes: para añadir los nombres de los ejes se utilizarán las siguientes sentencias:

  ```
  >set xlabel 'Tiempo de Pensar (s)'
  >set ylabel 'Frecuencia del Tiempo de Pensar'
  >replot
  ```

- Línea de Tiempo Medio de Pensar. Si *tm* es el tiempo medio de pensar de la transacción New-Order, se obtendrá una línea por medio de la sentencia:

  ```
  >set arrow 1 from tm, 0 to tm, fin
  >replot
  ```

  donde *fin* es la coordenada y del fin de línea.

Si se quieren eliminar los elementos dibujados, se deben seguir los pasos descritos en la gráfica 1.

Grafica 4. Esta gráfica se especifica en la cláusula 5.6.4 del estándar TPC-C. Representa la evolución del rendimiento en (tpmC) a lo largo del test. En el eje 'x' se representa el tiempo de test transcurrido en segundos. El eje 'y' representa el rendimiento ofrecido hasta ese punto.

El fichero que ha de utilizarse en este caso es `g4.dat`, y se encuentra en el directorio donde se llamó al programa y se ejecutó el recuento. Para visualizar la gráfica se utilizará la siguiente sentencia en Gnuplot:

```
gnuplot>plot 'g4.dat' with lines.
```

Una vez visualizada la gráfica se podrán añadir los elementos los elementos distitivos:

- Título de la gráfica: se puede titular la gráfica por medio de las sentencias:

```
>set title 'Evolución del Rendimiento'
>replot
```

- Nombre de los ejes: para añadir los nombres de los ejes se utilizarán las siguientes sentencias:

```
>set xlabel 'Tiempo Transcurrido (s)'
>set ylabel 'tpmC'
>replot
```

- Línea de Inicio de Periodo de Medida: Si si ti es el instante de inicio del periodo de medida, la línea se dibuja por medio:

```
>set arrow 1 from ti, 0 to ti, fin
>replot
```

  donde *fin* es la coordenada y del fin de línea.

- Línea de Fin de Periodo de Medida. Si *tf* el instante de fin del periodo de medida entonces, la línea se dibuja por medio:

```
>set arrow 2 from tf, 0 to tf, fin
>replot
```

  donde *fin* es la coordenada y del fin de línea.

- Líneas de Checkpoint. Si *tc(n)* es el instante de comienzo de un checkpoint y n el número del checkpoint, las líneas para cada uno de los checkpoints se pueden imprimir mediante:

```
>set arrow n+2 from tc, 0 to tc, fin
>replot
```

  donde *fin* es la coordenada y del fin de línea.

Si se quieren eliminar los elementos dibujados, se deben seguir los pasos descritos en la gráfica 1.

Cada una de las gráficas anteriores se puede almacenar en un fichero *.ps* de la siguiente forma:

```
>set terminal postscript
>set output 'fichero.ps'
>replot
```

En este momento habremos obtenido el fichero postscript: fichero.ps. Para poder volver a visualizar las gráficas en la pantalla:

```
>set terminal x11
>set output
>replot
```

De forma análoga se puede generar un fichero *.eps* cambiando la sentencia:

```
>set terminal postscript
```

por

```
>set terminal postscript eps
```

3.3. Ejecución manual del Monitor de Transacciones y del Emulador de Terminar Remoto

En el test no es posible ver la actividad de los Emuladores de Terminal Remoto (ETR) ya que estos no escriben los resultados por la pantalla. Si se desea ver el ciclo de ejecución de las transacciones y comprobar las pantallas que se generan en cada transacción, es posible ejecutar el Monitor de Transacciones (MT) y un ETR manualmente.

Si se realiza esta operación se sobreescribirán las bitácoras de rendimiento de un test anterior, por lo que ya no se podrá realizar un recuento de dicho test. Tampoco se podrá llevar a cabo un recuento del rendimiento ofrecido durante este ensayo.

Todo el proceso debe realizarse como usuario root.

1. Para arrancar el Monitor de Transacciones se deberá introducir la siguiente sentencia en un terminal:

   ```
   # tm
   ```

   aparecerá el siguiente símbolo:

   ```
   >>
   ```

2. Para lanzar un ETR se deberá introducir la siguiente sentencia en un otro terminal diferente:

   `# clien <ARG1><ARG2><ARG3><ARG4><ARG5>`

   donde:

   - `<ARG1>` es la llave que necesitan los clientes para crear el canal de comunicación con el Monitor de Transacciones: memoria compartida y semáforo de sincronización.
   - `<ARG2>` es el número de almacén al que pertenece el terminal. Tendrá que ser menor o igual al número de almacenes que contiene la base de datos.
   - `<ARG3>` es el número de distrito al que pertenece el terminal. Deberá estar comprendido entre 1 y 10.
   - `<ARG4>` es el número de almacenes que existen en la base de datos.
   - `<ARG5>` es el modo de ejecución: 0 la salida se envía a la pantalla, 1 la salida se envia a `/dev/null`.

3. Tras ejecutar esta sentencia se podrá visualizar la interacción entre el ETR y el MT.

   Si se desean lanzar más ETRs se deben seguir los mismos pasos mencionados anteriormente pero:

   - <ARG1>debe ser distinto para cada terminal.
   - <ARG2>, <ARG3>: por cada almacén solo pueden existir 10 terminales, uno por distrito. Por lo tanto la combinación de ambos ha de ser distinta para cada ETR ejecutado.

   .

4. Para parar la ejecución del conjunto de ETRs y del MT, se deben seguir los siguientes pasos:

   Ejecutar en un terminal:

   `# killall -TERM clien`

   esperar a que todos los ETR muestren el mensaje:

   `TERMINAL APAGADO`

   despues ejecutar:

   `# killall -TERM tm`

   Tras esto se apagará el Monitor de Transacciones.

3.4. Solución de Problemas

En este apartado se incluye una lista de soluciones de algunos problemas con los que usuario puede encontrarse al ejecutar el benchmark TPCC-UVA.

1. Al arrancar posgreSQL devuelve el siguiente mensaje:

   ```
   "Root" execution of the postgreSQL server is not permitted.

   The server must be started under an unprivileged userid to prevent
   a possible system security compromise. See the INSTALL file for
   more information on how to properly start the server.
   ```

   Ha intentado arrancar postgreSQL como usuario root. Pruebe a arrancarlo como usuario postgres.

2. Al arrancar posgreSQL devuelve el siguiente mensaje:

   ```
   postmaster does not find the database system.
   Expected to find it in the PGDATA directory "/usr/local/pgsql/data/",
   but unable to open file "/usr/local/pgsql/data/global/pg_control":
   Permission denied
   ```

   No ha arrancado postgreSQL como usuario autorizado. Pruebe a arrancarlo como usuario postgres.

3. Al ejecutar el benchmark aparece el siguiente mensaje:

   ```
   /usr/share/bin/bench: error while loading shared libraries:
   libecpg.so.3:
   cannot open shared object file: No such file or directory
   ```

   Puede deberse a una de las causas siguientes:

   - No se ha añadido la línea:
     `/usr/local/pgsql/lib`
     al fichero `/etc/ld.so.conf` o no se ha ejecutado el comando `ldconfig`.
   - Se ha utilizado el comando `su` para cambiar al usuario root sin el parametro '-'.

4. Al ejecutar el benchmark aparece el siguiente mensaje:

```
Error numero: -402
Could not connect to database 0V@8 in line 3364.
Se ha producido un error al comprobar la existencia de la
base de datos.
Error numero: -220
No such connection NULL in line 3371.
```

Arranque postgreSQL.

5. Al comenzar un test aparece el siguiente mensaje:

```
 ->Los clientes no disponen del fichero cons.dat.
No se puede continuar el test.
Continuar...
```

Se ha eliminado el fichero `/usr/share/var/tpcc/cons.dat`. Se deberá crear de nuevo una base de datos.

6. Al comenzar un test aparece el siguiente mensaje:

```
Error:
        No se puede abrir el fichero medida.log


Continuar ...
```

No ha ejecutado el benchmark como usuario root.

7. Al realizar un recuento de resultados aparece uno o más mensajes de la forma siguiente:

```
Error al abrir el fichero: /usr/share/var/tpcc/F_1_1.log
No such file or directory
```

Algunos de los ficheros de bitácora del ETR se han borrado. Se deberá volver a ejecutar el test.

8. Al realizar un recuento de resultados aparece el mensaje:

```
Error:
        No se puede abrir el fichero tm_delivery_tr.log

Abortando recuento ...

Continuar ...
```

El fichero /usr/share/var/tpcc/tm_deliver_tr.log se ha borrado. El test debe ejecutarse de nuevo.

9. Al realizar un recuento de resultados aparece el mensaje:

```
Error:
        No se puede abrir el fichero tm_delivery_res.log

Abortando recuento ...

Continuar ...
```

El fichero `/usr/share/var/tpcc/tm_deliver_res.log` se ha borrado. El test debe ejecutarse de nuevo.

En este capítulo se ha expuesto el manejo del benchmark TPCC-UVA. En el siguiente se explicarán características mas avanzadas de éste destinadas a aquellos programadores que quieran seguir desarrollando la actual implementación.

Capítulo 4

Manual del Programador

En este capitulo se profundiza en los aspectos relacionados con la compilación, la instalación y el empaquetado del benchmark TPCC-UVA, el sintonizado de la base de datos y la gestión de los recursos del sistema de los que se hacen uso. También se exponen una serie de puntos clave destinados a aquellos programadores que quieran seguir desarrollando el benchmark.

4.1. Ficheros fuente

Al descomprimir el paquete de fuentes `tpcc_uva-1.0-1.tar.gz` se pueden distinguir los directorios:

`SOURCES` : contiene los ficheros fuente de cada uno de los programas de que se compone el benchmark.

`include` : en este directorio se encuentra el fichero de cabecera `tpcc.h` del que hacen uso los programas.

`man` : contiene las páginas man de los programas que componen el benchmark en formato `.gz`.

`bin` : este directorio se utiliza durante la compilación de los ficheros fuente.

Los ficheros fuente almacenados en el directorio `SOURCES` son los siguientes:

`bench.c` : contiene el código del programa Controlador del Benchmark.

`tm.c` : contiene el código del programa Monitor de Transacciones.

`clitran.c` : contiene el código del programa Emulador de Terminal Remoto.

`check.c` : contiene el código del programa Controlador de Checkpoints.

`vacuum.c` : contiene el código del programa Controlador de Limpiezas.

En el paquete se encuentran, además, el fichero `Makefile` que contiene las instruciones de `make` para compilar, instalar y desinstalar los programas del benchmark.

El fichero de cabecera `tpcc.h` se definen una serie de constantes que determinan el funcionamiento del benchmark, que son:

- número de filas de cada tabla en la base de datos.
- número máximo de almacenes soportados por el benchmark.
- tiempos de teclado utilizados por los ETR.
- media de los tiempos de pensar utilizada por los ETR.
- tiempos del 90 % de las transacciones de cada tipo.
- intervalo entre checkpoints en la base de datos.

Además se definen las estructuras de datos utilizadas en las comunicaciones entre los Emuladores de Terminal Remoto y el Monitor de Transacciones y las funciones encargadas de la generación de datos aleatorios para la carga de la base de datos y la ejecución de las transacciones.

4.2. Fichero Makefile

El fichero se compone de cuatro secciones:

all : esta sección se encarga de compilar las fuentes del directorio `SOURCES` los ficheros binarios resultantes se escriben en el directorio `bin`. Se ejecuta cuando se llama al programa *make* de la forma:

```
# make
```

Antes de ser compilados, se comprueba la existencia de cada uno de los ficheros, interrumpiendo el proceso si alguno de ellos no existe.

install : en esta sección se crean los directorios `/usr/share/bin` y `/usr/share/var/tpcc`. Además, se copian los archivos binarios generados durante la compilación que se encuentran en el directorio `bin` al directorio `/usr/share/bin`. Se ejecuta cuando se llama al programa *make* de la forma:

```
# make install
```

En esta sección se comprueba la existencia de los archivos binarios en el directorio `bin`, al igual que en la sección anterior, si alguno de los ficheros no existe, se interrumpe el proceso.

uninstall : en esta sección se borran los ficheros ejecutables del benchmark del directorio `/usr/share/bin`. El contenido del directorio `/usr/share/var/tpcc` no se elimina. Se ejecuta cuando se llama al programa *make* de la forma:

```
# make uninstall
```

clean : elimina los ficheros ejecutables creados en la etapa de compilación del directorio `bin`. Se ejecuta cuando se llama al programa *make* de la forma:

```
# make clean
```

4.3. Compilación de las fuentes

Los ficheros fuente `bench.c`, `tm.c`, `check.c` y `vacuum.c`, contienen sentencias escritas en lenguaje SQL embebido, por lo que tienen que ser preprocesados por un programa que transforme esas sentencias a funciones de postgreSQL. Ese programa preprocesador es el *ecpg* [27] que se incluye con postgreSQL-7.1.3.

El preproceso se realiza de la siguiente forma:

```
# ecpg fichSQL.c -o fichECPG.c
```

la opción `-o` indica que `fichECPG.c` es el fichero donde queremos que se produzca la salida.

En el apándice C se profundiza en el funcionamiento del preprocesador *ecpg*.

La compilación se efectúa mediante:

```
# gcc -I /usr/local/pgsql/include -o binario fichECPG.c
-L /usr/local/pgsql/lib -lm -lpq -lecpg
```

- `-I` se utiliza para incluir los ficheros de cabecera de postgreSQL.
- `-o` se utiliza para indicar el nombre del ejecutable que se quiere crear.
- `-L` se utiliza para indicar el directorio de librerías de postgreSQL.

- `-lm` se utiliza para enlazar con la librería *math*.
- `-lpq, -lecpg` se utilizan para enlazar con las librerías de ecpg y postgreSQL.

Nota: no se han utilizado ninguna opción de optimización en la compilación de las fuentes debido a que algunas de las optimizaciones bloquean partes fundamentales del benchmark, como por ejemplo los bucles `while` vacíos, que se utilizan para interrumpir un proceso hasta la llegada de una señal.

En el fichero fuente `clitran.c` no hay sentencias SQL, por lo que no es necesario preprocesarlo con *ecpg*. La compilación se realiza mediante:

```
# gcc fichero.c -o binario -lm
```

donde el significado del las opciones `-o` y `-lm` es el mismo que en el caso anterior.

4.4. Empaquetado del benchmark utilizando RPM

El conjunto de programas que componen el benchmark pueden empaquetarse fácilmente en formato el *rpm* de Linux Red Hat. Para ello, se debe escribir un archivo con la extensión `.spec` y situarlo en el directorio `/usr/src/redhat/SPECS`. Los archivos `.spec` son imprescindibles para construir un paquete. Contienen una descripción del software y las instrucciones necesarias para construir el paquete, además de una lista de todos los ficheros instalados.

La información que debe contener el fichero `.spec` para el benchmark es la siguiente:

```
Summary: Implementación del benchmark TPC-C, Departamento de Informática,
Universidad de Valladolid.
Name: tpcc_uva
Version: 1.0
Release: 1
Source: tpcc_uva-1.0-1.tar.gz
Copyright: GPL
Group: Utilities/System
%description
Implementación del benchmark TPC-C (http://www.tpc.org/tpcc/default.asp).
Grupo de Arquitectura de Computadores. Dpto. de Informática. Universidad
de Valladolid. http://www.info.uva.es/ diego
%prep
tar xzvf ../SOURCES/tpcc_uva-1.0-1.tar.gz
%build
```

```
cd tpcc_uva.1.0-1
make
%install
cd tpcc_uva.1.0-1
%files
/usr/share/bin/bench
/usr/share/bin/tm
/usr/share/bin/clien
/usr/share/bin/check
/usr/share/bin/vacuum
/usr/share/man/man1/bench.1.gz
/usr/share/man/man1/tm.1.gz
/usr/share/man/man1/clien.1.gz
/usr/share/man/man1/check.1.gz
/usr/share/man/man1/vacuum.1.gz
```

En este fichero se pueden distinguir los siguientes campos:

Summary Descripción en una sola línea del paquete.

Name El nombre del fichero rpm que se va a generar.

Version El número de versión del paquete.

Release El número de publicación para un paquete dentro de un mismo número de versión.

Source El nombre del fichero que contiene las fuentes.

Copyright Indica el tipo de copyright al que esta sometido el paquete.

Group Enmarca al paquete dentro de la jerarquía de rpm.

%description Es un campo multilínea que proporciona una descripción comprensible del paquete.

%prep Debajo de esta etiqueta se puede crear un script simple para el shell *sh* para desempaquetar las fuentes y prepararlas antes de la compilación. Se debe tener en cuenta que es en el directorio `/usr/src/redhat/BUILD/` donde se ejecuta el script.

%make Debajo de esta etiqueta se deben incluir los comandos necesarios para compilar el software. Se debe tener en cuenta que al comienzo de esta etapa el directorio actual es `/usr/src/redhat/BUILD`.

%install Se deben incluir en este campo los comandos necesarios para instalar el software. Se debe tener en cuenta que al comienzo de esta etapa el directorio actual es `/usr/src/redhat/BUILD`.

%files En esta sección se incluye la lista de los archivos que se instalan con el paquete.

Una vez que se ha creado el fichero `.spec`, se puede obtener los paquetes con las fuentes y con los binarios ejecutando:

```
# rpm -ba /usr/src/redhat/SPECS/fichero.spec
```

El paquete con los ejecutables se creara en:

```
/usr/src/redhat/RPMS/i386/
```

y el paquete con las fuentes en:

```
/usr/src/redhat/SRPMS/
```

Si se desea obtener más información acerca de la creación de paquetes `rpm` se puede consultar la URL:

```
http://www.rpm.org/RPM-HOWTO/
```

4.5. Sintonizado de postgreSQL

La configuración de postgreSQL se realiza a través del fichero de configuración `/usr/local/pgsql/data/postgresql.conf` [25]. Los cambios realizados en este fichero tienen efecto cuando se reinicia postgreSQL o se le envía la señal SIGHUP como usuario postgres de la forma:

```
# killall -HUP postmaster
```

Durante la configuración del entrono y la instalación del benchmark se modifican varios de los parámetros de este fichero que afectan e rendimiento durante el test. Para más información sobre el sintonizado de postgres se recomienda consultar el manual de administrador de postgreSQL-7.1.3. disponible en la URL:

```
ftp.es.postgresql.org/postgresql/doc/7.1/admin.pdf
```

y en el CD adjunto.

4.5.1. Checkpoints

PostgreSQL, para aumentar la velocidad de las consultas, mantiene unos ficheros secuenciales (WAL Files [25]) que contienen las modificaciones que se han

realizado sobre la base de datos. De esta forma, cada vez que se produce una modificación, se escribe en los ficheros secuenciales en vez de realizarla directamente en la base de datos. El acceso a un fichero secuencial es más rápido que la modificación de la estructura de datos que mantiene las tablas.

Tras un checkpoint se libera la información de los ficheros secuenciales trasladando las modificaciones a la base de datos. PostgreSQL realiza un checkpoint automático si los ficheros secuenciales se han llenado o si el periodo de tiempo máximo entre dos checkpoints automáticos ha expirado.

Para cumplir las especificaciones del TPC-C, el Controlador de Checkpoints realiza checkpoints periodicos sobre la base de datos, cada 30 minutos.

Es necesario evitar los checkpoints que postgreSQL realiza automáticamente, ya que pueden suceder entre dos checkpoints forzados por el Controlador. Para ello se han modificado los siguientes parámetros en el fichero de configuración de postgreSQL:

`wal_files` indica el número de ficheros secuenciales que se crean al arrancar postgreSQL. Se ha aumentado este parámetro al valor 10.

`checkpoint_segments` indica el número de ficheros secuenciales que deben llenarse antes de hacer un checkpoint. Se ha aumentado al valor 10.

`checkpoint_timeout` indica el tiempo máximo, expresado en segundos entre dos checkpoints automáticos. Se ha aumentado al valor 3600.

Con la configuración anterior, se reduce en gran medida la posibilidad de que postgreSQL realice un checkpoint antes que el Controlador de Checkpoints.

4.5.2. Volcado al disco

Por defecto postgreSQL, en cada transacción, utiliza en numerosas ocasiones la llamada al sistema *fsync()* [25] para asegurar que las actualizaciones de la base de datos se han escrito en el disco en vez de almacenarse en la cache del kernel. Esto aumenta las posibilidades de recuperar los datos después de un fallo del sistema operativo o del hardware. Sin embargo, esta operación ralentiza a postgreSQL, ya que cada vez que se llama a *fsync()* tiene que esperar a que el sistema operativo vuelque los datos al disco.

PostgreSQL permite desactivar las llamadas a *fsync()*, lo que aumentaría de forma notable el rendimiento del sistema. Sin embargo, si se produce un fallo en el sistema, los resultados de las ultimas transacciones confirmadas pueden perderse en parte o por completo; en el peor de los casos la consistencia de los datos puede ser irrecuperable.

Debe deshabilitarse esta opción si el sistema operativo, el hardware y la alimentación están lo suficientemente protegidas ante fallos.

En el fichero `postgresql.conf` esta opción se desactiva modificando la línea:

```
#fsync = true
```

por

```
fsync = off
```

4.6. Modificaciones del software

Esta sección pretende orientar al programador hacia los puntos clave a los que se debe dirigir si desea modificar alguna de las características principales del benchmark TPCC-UVA.

4.6.1. Número máximo de almacenes admitidos por el benchmark

Para modificar el número máximo de almacenes con los que puede trabajar el benchmark, se deberá modificar la constante `NUM_MAX_ALM` definida en el fichero de cabecera `tpcc.h`. El valor actual de esta constante es 100.

Además se recomienda tener en cuenta las limitaciones en cuanto al los recursos IPC mencionadas en el apartado 4.7.

4.6.2. Tiempos de espera definidos para los ETRs

Los tiempos de espera definidos en las cláusulas del TPC-C para los Emuladores de Terminal Remoto pueden ajustarse modificando las constantes definidas en el fichero de cabecera `tpcc.h`. Los requisitos acerca de los tiempos de espera se especifican en la cláusula 5.2.5 del TPC-C.

- Tiempos de Teclado.

  Los tiempos de teclado para cada tipo de transacción están definidos mediante las constantes:

  - `TT_NEW_ORDER`, fijado a 18 (s).
  - `TT_PAYMENT`, fijado a 3 (s).

- `TT_OSTATUS`, fijado a 2 (s).
- `TT_DELIVERY`, fijado a 2 (s).
- `TT_STOCK_LEVEL`, fijado a 2 (s).

- Tiempos de Pensar.

  Las medias de las distribuciones aleatorias del tiempo de pensar para cada transacción están definidas mediante las constantes:

  - `TMP_NEW_ORDER`, fijada a 12 (s).
  - `TMP_PAYMENT`, fijada a 12 (s).
  - `TMP_OSTATUS`, fijada a 10 (s).
  - `TMP_DELIVERY`, fijada a 5 (s).
  - `TMP_STOCK_LEVEL`, fijada a 5 (s).

4.6.3. Tiempo de respuesta del 90 % de las transacciones

Los tiempos de respuesta por debajo de los cuales deben haberse completado el 90 % de las transacciones se definen en el fichero de cabecera `tpcc.h` mediante las constantes:

- `TR90thNO`, fijada a 5 (s).
- `TR90thP`, fijada a 5 (s).
- `TR90thOS`, fijada a 5 (s).
- `TR90thD`, fijada a 5 (s).
- `TR90thSL`, fijada a 20 (s).

Los requisitos en cuanto a los tiempos de respuesta se especifican en la cláusula 5.2.5 del TPC-C.

4.6.4. Escalado de la base de datos

En el fichero de cabecera `tpcc.h` se definen una serie de constantes que determinan el escalado de la base de datos, como es, por ejemplo, el número de filas de la tabla `item` o el número de clientes por cada distrito.

4.6.5. Definición de las tablas de la base de datos

La función encargada de crear las tablas en la base de datos según las especificaciones del estándar TPC-C se define en el fichero fuente del Controlador del Benchmark: `bench.c`. La función se llama `creartablas()`.

4.6.6. Perfiles de transacción

Los perfiles de las transacciones definidas en el estándar TPC-C se implementan mediante las funciones:

- `trans_new_order()` para la transacción New-Order.
- `trans_payment()` para la transacción Payment.
- `trans_ostatus()` para la transacción Order-Status.
- `trans_delivery()` para la transacción Delivery.
- `trans_stock_level()` para la transacción Stock-Level.

declaradas en el fichero fuente del Monitor de Transacciones: `tm.c`. Si se desea modificar la implementación de los perfiles, se deberán modificar las funciones anteriores.

4.6.7. Creación de indices en las tablas

En la versión actual del benchmark no se crea ningún índice en un campo que no sea *primary key*. Los índices con los campos que constituyen una *primary key* los genera automáticamente postgreSQL al declararles como *primary key*.

Si se desea crear índices adicionales en la base de datos, puede utilizarse el monitor interactivo que incluye postgreSQL, `psql` (ver apéndice C) o puede reescribirse la definición de las tablas de la función `creartablas()` del fichero fuente del Controlador del Benchmark: `bench.c`.

4.7. Recursos IPC

Las comunicaciones entre la población de ETRs y el MT se implementan a través de los mecanismos de comunicación entre procesos IPC, de UNIX System

V [15]. A continuación se detallan las limitaciones de estos recursos y algunas técnicas para su manejo.

4.7.1. Limitaciones de los recursos IPC

Cada sistema operativo tiene sus propias limitaciones en cuanto al número máximo de recursos IPC disponibles. En los sistemas linux, se definen una serie de constantes que determinan el número total de recursos disponibles.

Memoria compartida

Las constantes que definen las limitaciones de la memoria compartida se declaran en el fichero:

`/usr/include/linux/shm.h`

Algunas de las constantes que se definen en este fichero son:

`SHMMNI` Número máximo de segmentos de memoria compartida en todo el sistema.

`SHMSEG` Número máximo de segmentos de memoria compartida por proceso.

`SHMMAX` Tamaño máximo, expresado en bytes, de un segmento de memoria compartida.

`SHMMIN` Tamaño mínimo, expresado en bytes, de un segmento de memoria compartida.

Semáforos

Las constantes que definen las limitaciones de los semáforos se declaran en el fichero:

`/usr/include/linux/sem.h`

Algunas de las constantes que se definen en este fichero son:

`SEMMNI` Número máximo de conjuntos de semáforos en todo el sistema.

`SEMMNS` Número máximo de semáforos en todo el sistema.

`SEMMSL` Número máximo de semáforos por conjunto de semáforos.

Colas de mensajes

Las constantes que definen las limitaciones de las colas de mensajes se declaran en el fichero:

`/usr/include/linux/msg.h`

Algunas de las constantes que se definen en este fichero son:

`MSGMAX` Tamaño máximo, en bytes, que se puede enviar a través de la cola.

`MSGMNB` Número máximo de mensajes en una cola en particular.

`MSGTQL` Número máximo de mensajes en todo el sistema.

4.7.2. Control de los recursos IPC desde la línea de comandos

Los programas estándar *ipcs* e *ipcrm* [16] permiten controlar los recursos IPC que gestiona el sistema. Ipcs se utiliza para ver los mecanismos que están asignados y a quién, junto con información estadística. Ipcrm se emplea para liberar un mecanismo asignado y dejar libre la entrada que ocupa. Estos dos programas son bastante útiles para depurar programas que se valen de los mecanismos IPC, como es el caso del benchmark.

La forma de emplear *ipcs* es la siguiente:

```
# ipcs [ opciones ]
```

Si no se especifica ninguna opción, *ipcs* muestra un resumen de la información administrativa que se almacena para los semáforos, memoria compartida y colas de mensajes que hay asignados. Las principales opciones son:

- `-q` Muestra información de las colas de mensajes que hay activas.
- `-m` Muestra información de los segmentos de memoria compartida que hay activos.
- `-s` Muestra información de los semáforos que hay activos.
- `-b` Muestra una información completa sobre los tres tipos de mecanismos IPC que hay activos.

La forma de emplear *ipcrm* es la siguiente:

```
# ipcrm [ opciones ]
```

Algunas de las opciones disponibles son:

- `-q msqid` Borra la cola de mensajes cuyo identificador coincide con `msqid.`
- `-m shmid` Borra la zona de memoria compartida cuyo identificador coincide con `shmid`.
- `-s semid` Borra el semáforo cuyo identificador coincide con `semid`.

4.8. Comprobación del tamaño de la base de datos

La base de datos del benchmark crece a medida que se desarrolla el test de rendimiento. El tamaño inicial de la base de datos es de 137Mb por almacén configurado.

Para comprobar el tamaño de la base de datos después de un test, se puede hacer uso del Monitor Interactivo de Base de Datos que proporciona postgreSQL: el `psql`.

Los pasos que habrá que seguir son:

- Arrancar psql conectando con la base de datos 'tpcc' mediante la siguiente sentencia:

  ```
  # psql tpcc postgres
  tpcc=# select oid from pg_database where datname = 'tpcc';
  ```

  La salida de esta sentencia será:

  ```
  oid
  -----
  1257032
  (1 row)
  ```

  El número *oid* identifica al directorio donde se encuentra la base de datos.

- Salir de psql mediante:

  ```
  tpcc=# \q
  ```

- Como usuario *postgres* introducir la siguiente sentencia:

  ```
  # du -h /usr/local/pgsql/data/base/oid
  ```

  donde *oid* es el número obtenido.

Tras esto, por pantalla aparecerá el tamaño actual de la base de datos:

```
147M  /usr/local/pgsql/data/base/1257032
```

Para más información acerca del monitor interactivo consular el apéndice C de esta memoria o el manual de referencia del postgreSQL que se incluye en el CD adjunto.

En este capítulo se han expuesto algunas características avanzadas del TPCC-UVA, destinadas a los programadores que deseen desarrollar la implementación actual. En el siguiente, se analizarán una serie de resultados obtenidos con este benchark.

Capítulo 5

Análisis de resultados.

Este capítulo incluye una serie de resultados obtenidos con el benchmark TPCC-UVA en máquinas uniprocesador. El estudio de los resultados permitirá realizar un análisis de las limitaciones del benchmark así como de las posibles soluciones.

5.1. Resultados.

A continuación se exponen los resultados completos de una serie de tests realizados con el fin de atestiguar el funcionamiento del benchmark TPCC-UVA. En primer lugar, en las secciones 5.1.1, 5.1.2 y 5.1.3 se detallan las características y las salidas del benchmark de tres tests, ejecutados en diferentes sistemas uniprocesador. Los tres tests se han ejecutado con un periodo de medida de dos horas y sin realizar limpiezas en la base de datos. Seguidamente, en la sección 5.1.4 se hará una valoración conjunta de los resultados de los tests. Por ultimo, en la sección 5.1.5 se incluyen los resultados de dos tests de ocho horas de duración, y se analizará la necesidad de realizar limpiezas periódicas en la base de datos en tests de larga duración, a fin de evitar la disminución del rendimiento provocado por postgreSQL.

5.1.1. Test 1

Las características del sistema en el que se ha ejecutado este test con el benchmark TPCC-UVA son:

- Procesador: Intel Pentium II 266 Mhz.
- Memoria RAM: DIMM 64 Mb 66 Mhz.
- Memoria Caché: L1 32k, L2 512k.
- Disco Duro: Compaq 3Gb.
- Sistema operativo: Linux Red Hat 7.2.
- kernel: 2.4.7-10.
- Compilador: gcc versión 2.96.
- Preprocesador SQL: ecpg versión 2.8.0.
- RDBMS: postgreSQL 7.1.3.

Para la compilación del benchmark no se ha utilizado ningún flag de optimización, ya que determinadas partes del programa se ven afectadas por las optimizaciones de gcc. En los test sucesivos la compilación se ha realizado de la misma forma.

La prueba se realizó el día 26 de Mayo de 2002 a las 13:24:57. Los parámetros del test fueron los siguientes:

- Número de almacenes: 1.
- Número de Emuladores de Terminal Remoto por almacén: 10.
- Periodo de rampa: 20 m.
- Periodo de medida: 120 m.
- Sin limpiezas en la base de datos.

El test se realiza con 10 terminales por almacén como indica el estándar. La finalidad del periodo de rampa es esperar a que el rendimiento del sistema se estabilize para comenzar el periodo de medida.

Los resultados obtenidos en el sistema con la configuración descrita ha sido los siguientes:

```
RENDIMIENTO OBTENIDO: 11.633 tpmc para 1 almacenes.

 3189 Transacciones en total.

TRANSACCIONES NEW ORDER:
  1396 Transacciones en periodo de medida. 1634 Totales.
  Porcentaje: 43.775%
  Porcentaje bien hechas : 91.762%
  Tiempo de respuesta: min/med/max/90th: 0.194/2.009/13.200/4.520
  Porcentaje de transacciones rechazadas: 1.074% .
  Número medio de artículos por orden: 9.917 .
  Porcentaje de artículos remotos: --- (Sólo un almacén).
  Tiempo de pensar min/med/max: 0.000/12.258/96.000

TRANSACCIONES PAYMENT:
  1381 Transacciones en periodo de medida. 1630 Totales.
  Porcentaje: 43.305%
  Porcentaje bien hechas : 93.193%
  Tiempo de respuesta: min/med/max/90th: 0.030/1.431/16.092/3.880
  Porcentaje de transacciones remotas: --- (Sólo un almacén)
  Porcentaje de clientes seleccionados por C\_ID: 41.419% .
  Tiempo de pensar min/med/max: 0.000/12.224/85.000

TRANSACCIONES ORDER STATUS:
  136 Transacciones en periodo de medida. 161 Totales.
  Porcentaje: 4.265%
  Porcentaje bien hechas : 92.647%
  Tiempo de respuesta: min/med/max/90th: 0.113/1.826/8.805/4.000
  Porcentaje de clientes seleccionados por C\_ID: 41.176% .
  Tiempo de pensar min/med/max: 0.000/9.684/48.000

TRANSACCIONES DELIVERY:
  137 Transacciones en periodo de medida. 162 Totales.
  Porcentaje: 4.296%
  Porcentaje bien hechas : 100.000%
  Tiempo de respuesta: min/med/max/90th: 0.000/0.000/0.000/0.000
  Porcentaje ejecucion < 80s : 100.000%
  Tiempo de ejecucion min/med/max: 2.243/5.018/15.133
  N° de distritos saltados: 0 .
  Porcentaje de distritos saltados: 0.000%.
  Tiempo de pensar min/med/max: 0.000/4.803/24.000

TRANSACCIONES STOCK LEVEL:
  139 Transacciones en periodo de medida. 165 Totales.
  Porcentaje: 4.359%
  Porcentaje bien hechas : 100.000%
  Tiempo de respuesta: min/med/max/90th: 1.498/5.703/16.428/9.440
```

```
  Tiempo de pensar min/med/max: 0.000/4.849/24.000

Checkpoints con mayor duración:
Hora de comienzo           Tiempo desde el Inicio (s)  Duración (s)
 Mon May 27 14:28:07 2002    3009.146000                16.228000
 Mon May 27 13:57:59 2002    1201.052000                8.097000
 Mon May 27 14:58:24 2002    4826.466000                7.554000
 Mon May 27 15:28:30 2002    6634.014000                6.074000

>> TEST PASADO
```

A continuación se explica el significado de los datos anteriores para después hacer un análisis de los mismos:

- Rendimiento obtenido: representa la tasa de rendimiento en (tpmC) ofrecida por el sistema durante el test. Se corresponde con el número de transacciones New-Order ejecutadas por minuto durante el periodo de medida.
- Número total de transacciones ejecutadas durante el periodo de medida.

Para cada tipo de transacción se muestra:

- Número de transacciones durante el periodo de medida y durante el test completo.
- Porcentaje de transacciones de ese tipo con respecto al total. Para que sea considerado como válido, las transacciones Payment deben superar el 43 % y las transacciones Order-Status, Delivery y Stock-Level el 4 %. El resto se deja para las transacciones New-Order. (ver cláusula 5.2.3 del estándar TPC-C).
- Porcentaje de transacciones bien hechas: es el porcentaje de transacciones cuyo tiempo de respuesta de transacción es menor que el especificado por el estándar TPC-C, que son:
  - 5 segundos para las transacciones New-order, Payment, Order-Status y Delívery.

    Nota: En el caso de la transacción Delivery este tiempo se refiere al tiempo de encolado, no de ejecución (ver perfil de transacción Delivery de la cláusula 2.7 estándar TPC-C).
  - 20 segundos para la transacción Order-Status.

Nota: El tiempo de respuesta de transacción se define como el tiempo transcurrido desde que el terminal envía la transacción hasta que recibe los resultados. (ver definición de tiempo de respuesta de transacción en la cláusula 5.3.4 del estándar TPC-C).

Para que el rendimiento obtenido sea válido según las especificaciones del estándar TPC-C, el 90 % de las transacciones han de tener un tiempo de respuesta de transacción inferior a los anteriores.

- Tiempo de Respuesta de Transacción mínimo, medio, máximo, y del 90 % de las transacciones, es decir tiempo por debajo del cual se ejecutan el 90 % de las transacciones.

  Para que el rendimiento obtenido sea válido según las especificaciones del estándar TPC-C, el tiempo de respuesta del 90 % de transacciones ha de ser menor que los anteriormente citados.

- Tiempos de pensar mínimo, medio, y máximo. Estos tiempos se simulan en los ETR para imitar el tiempo en el que el usuario toma decisiones. Para cada tipo de transacción, el estándar TPC-C especifica que los tiempos medios mínimos de pensar de los terminales han de ser:

  - Para las transacciones New-Order y Payment, 12 segundos.
  - Para la transacción Order-Status, 10 segundos.
  - Para las transacciones Delivery y Stock-Level, 5 segundos.

Además para la transacción New-Order se muestra:

- Porcentaje de transacciones canceladas (rollback). El estándar TPC-C especifica que este porcentaje ha de ser del 1 %, permitiéndose una variación del $\pm 0,1$ % (ver cláusula 2.4.1.4 del estándar).

- Número medio de artículos por cada orden. El estándar TPC-C especifica que este número medio ha de ser 10, permitiéndose una variación de $\pm 0,5$ (ver cláusula 2.4.1.3 del estándar).

- Porcentaje de artículos remotos. El estándar TPC-C especifica que este porcentaje ha de ser del 1 %, permitiéndose una variación de $\pm 0,05$ % (ver cláusulas 2.4.1.5 y 2.4.1.8 del estándar).

Para la transacción Payment se muestra:

- Porcentaje de transacciones Payment remotas. El estándar TPC-C especifica que este porcentaje ha de ser del 15 %, permitiéndose una variación de ± 1 % (ver cláusula 2.5.1.2 del estándar).

- Porcentaje de clientes seleccionados por C_ID (identificador de clientes). El estándar TPC-C especifica que este porcentaje ha de ser del 40 %, permitiéndose una variación de ±3 % (ver cláusulas 2.5.1.2 del estándar).

Para la transacción Order-Status se muestra:

- Porcentaje de clientes seleccionados por C_ID (identificador de clientes). El estándar TPC-C especifica que este porcentaje ha de ser del 40 %, permitiéndose una variación de ±3 % (ver cláusula 2.6.1.2 del estándar).

Para la transacción Delivery se muestra:

- Porcentaje de transacciones con un tiempo de ejecución menor de 80 segundos. El estándar TPC-C especifica que este porcentaje ha de ser por lo menos del 90 % (ver cláusula 5.2.5.7 del estándar)

  Nota: El tiempo de ejecución para la transacción Delivery se define como el tiempo transcurrido desde que el Emulador de Terminal Remoto envía la transacción hasta que el Monitor de transacciones finaliza su ejecución. (ver definición de tiempo de ejecución en la cláusula 2.7.2.2 del estándar TPC-C).

- Tiempo de ejecución mínimo, medio, máximo, y del 90 % de las transacciones (es decir, tiempo por debajo del cual se ejecutan el 90 % de las transacciones). El estándar especifica que el tiempo de ejecución del 90 % de transacciones Delivery ha de ser menor de 80 segundos.

- Número y porcentaje de distritos saltados como resultado de no encontrarse ninguna orden pendiente de reparto. Este dato sirve para informar si el número de distritos saltados representa más del 1 % del total de distritos procesados o que esta circunstancia ha ocurrido en más de una ocasión, como se especifica en la cláusula 2.7.4.2 del estándar TPC-C.

Además de lo anterior se muestra:

- Checkpoints de mayor duración:

  Se muestra información acerca de los cuatro *checkpoints* de mayor duración ordenados por orden descendente. Este benchmark ejecuta checkpoints cada 30 minutos comenzando en el inicio del periodo de medida. En la cláusula 5.5.2.2 del estándar TPC-C se describe el termino *checkpoint* y las consideraciones acerca de su ejecución durante el test de rendimiento.

  La información que se muestra es:

  - Fecha y hora de comienzo del checkpoint.
  - Tiempo de test transcurrido al comienzo del checkpoint, expresado en segundos.
  - Duración en segundos del checkpoint.

- El mensaje `TEST PASADO` indica que el test han cumplido los requisitos en cuanto a los tiempos respuesta de las transacciones. De no haberse cumplido, se muestra el mensaje `TEST FALLIDO`.

De los datos anteriores, dos son los que verdaderamente determinan el rendimiento y la eficiencia del sistema: el número de transacciones por minuto (tpmC), y los tiempos de respuesta de transacción. El resto determina si la carga de trabajo se ha ejecutado según las especificaciones del estándar TPC-C.

De los resultados anteriores se pueden sacar las siguientes conclusiones:

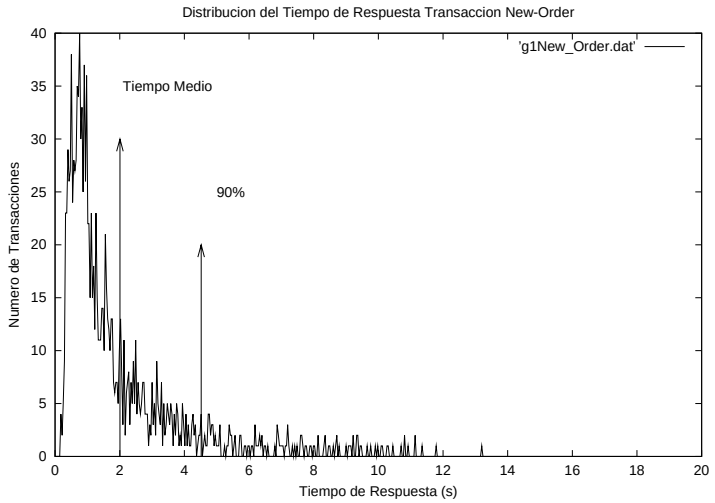
- La tasa de rendimiento ha sido de 11,633 tpmC.
- Se han cumplido los porcentajes de la mezcla de transacciones.
- En las transacciones New-Order, Payment y Stock-Level, aunque los porcentajes y los tiempos de respuesta para el 90 % de las transacciones son válidos, se aproximan demasiado a los límites permitidos. Por ejemplo, para la transacción New-Order estos valores son 91,762 % y 4,520s respectivamente.
- No se muestran los porcentajes de artículos remotos en la transacción New-Order y el porcentaje de transacciones Payment remotas ya que el test sólo se ha ejecutado con un único almacén.
- Los tiempos medios de pensar para las transacciones Order-Status, Delivery y Stock-Level no han alcanzado los especificados por el estándar ya que no se han ejecutado las suficientes transacciones como para que se alcance el valor medio de la distribución aleatoria con la que se generan. No obstante, las desviaciones no superan el 5 % por lo que el impacto sobre el rendimiento es mínimo.
- Los tiempos de respuesta de la transacción Delivery son cero, lo que indica que su valor ha sido menor que la precisión mínima utilizada por el programa. Esto se debe a que este tiempo se corresponde con los tiempos de encolado, que en este benchmark son insignificantes.
- El porcentaje de transacciones Delivery ejecutadas en menos de 80s cumple sobradamente con las especificaciones, además cabe destacar que el tiempo máximo de ejecución ha sido de 15.133 muy por debajo del máximo especificado.

- El resto de valores cumplen las especificaciones del estándar.

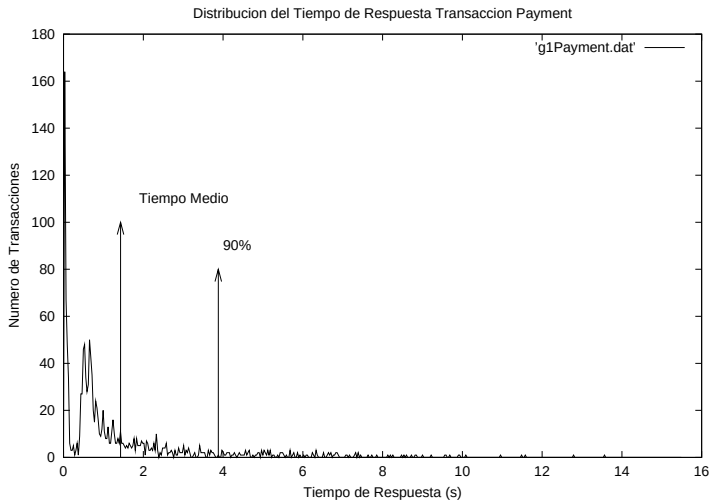
A continuación se detallan las gráficas de resultados que se han generado.

Gráficas de distribución del tiempo de respuesta

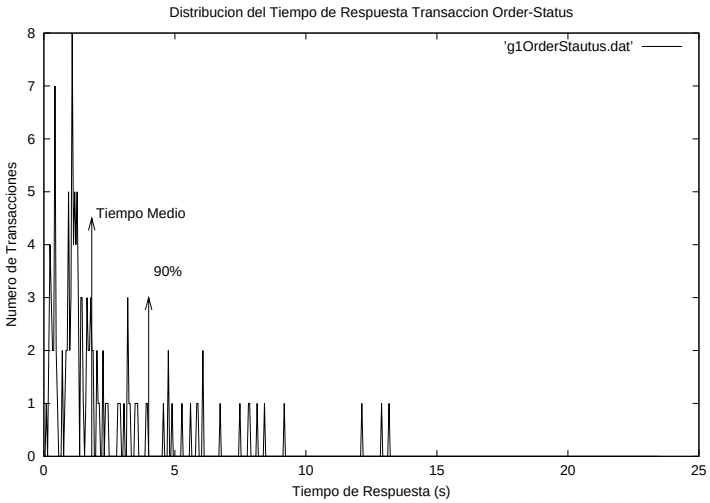
Estas gráficas se definen en las cláusulas 5.6.1 del estándar TPC-C. Se obtiene una gráfica por cada tipo de transacción, en la que se representa el número de transacciones completadas con un determinado tiempo de respuesta. La gráfica de la transacción Delivery no se ha incluido debido a que los tiempos de respuesta son insignificantes.



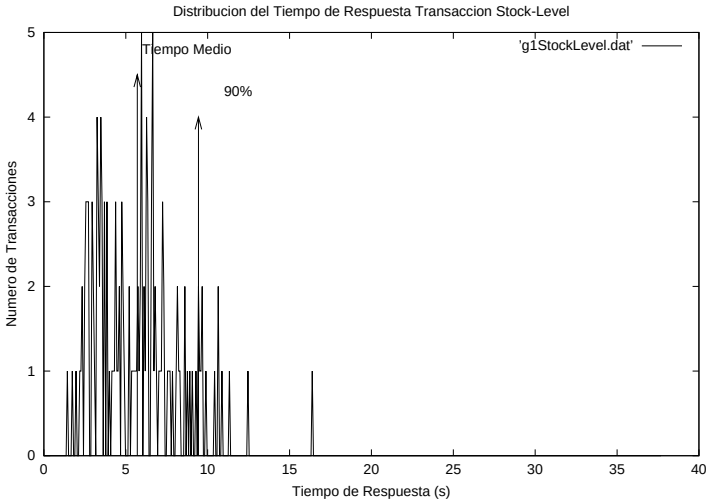
Distribución del tiempo de respuesta para la transacción New Order



Distribución del tiempo de respuesta para la transacción Payment



Distribución del tiempo de respuesta para la transacción Order Status

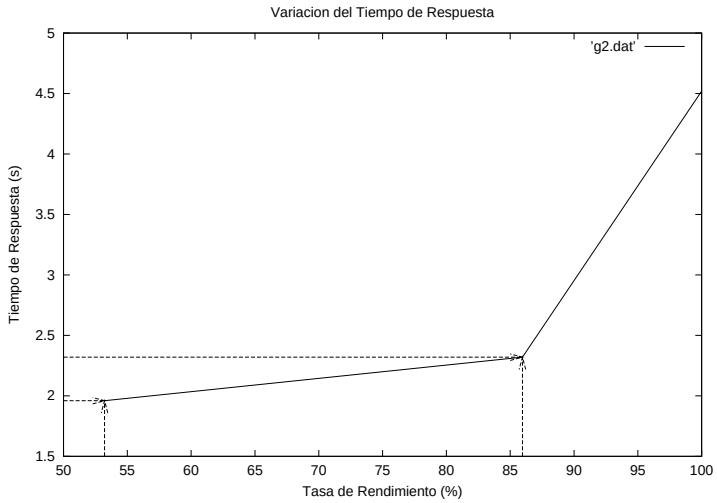


Distribución del tiempo de respuesta para la transacción Stock level

En las gráficas anteriores se puede comprobar la separación entre los tiempos medios de respuesta y el tiempo de respuesta para el 90 % de las transacciones. Además, cabe resaltar que los tiempos medios son menores que los tiempos del 90 %, como requiere el estándar. En la gráfica de la transacción Stock-Level se puede apreciar que ambos tiempos son mayores que los del resto de transacciones, ya que ésta incluye operaciones de base de datos con mayor tiempo de ejecución.

Gráfica de variación del tiempo de respuesta

Esta gráfica se define en la cláusula 5.6.2 del estándar y representa la variación del tiempo de respuesta para el 90 % de las transacciones New-Order con respecto al rendimiento medido.



Variación del Tiempo de Respuesta para la Transacción New Order.

La gráfica se compone de tres puntos. El primero corresponde con el tiempo de respuesta para el rendimiento máximo obtenido en el test (100 %). Los restantes puntos se han obtenido ejecutando tests de rendimiento de media hora de duración con 8 ETRs para el punto del 86 % y con 5 ETRs para el del 54 %.

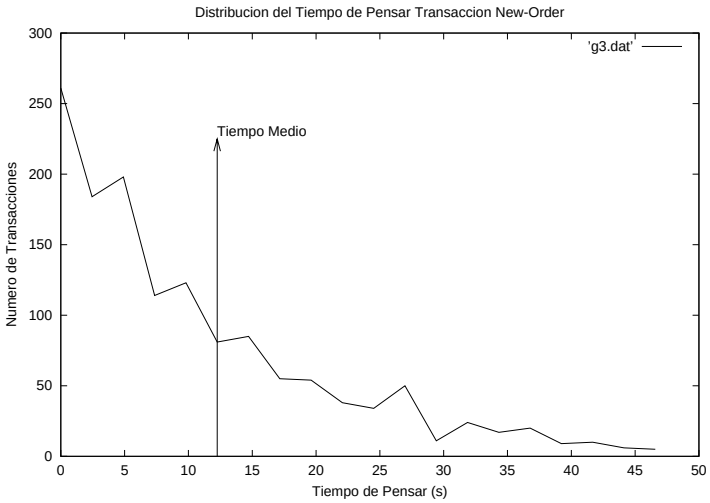
Se puede observar cómo al suprimir la actividad de dos terminales por almacén el tiempo de respuesta ha disminuido notablemente. Esto es debido al mecanismo que utiliza el Monitor de Transacciones para asegurar el aislamiento entre transacciones. El MT recibe las solicitudes de transacción por una cola de mensajes y las ejecuta en serie según su orden de llegada, lo que provoca que las transacciones encoladas esperen a la ejecución de las que están por delante. El hecho de reducir la población de terminales disminuye la probabilidad de que una transacción espere en la cola y por otro lado disminuye el número de transacciones en espera.

Además, se comprueba que eliminando tres ETR más por almacén la disminución del tiempo de respuesta es menos acusada. Esto ratifica la influencia de la cola de mensajes puesto que, al reducir la probabilidad de que las transacciones esperen en la cola, el tiempo de respuesta se compondrá en su mayor parte del tiempo de ejecución de la transacción, que no depende de la carga de la cola ni, por lo tanto, del número de terminales.

Como conclusión, el tiempo de espera en la cola depende directamente de la población de terminales, mientras que el tiempo de ejecución depende exclusivamente de la propia transacción, y ya que el tiempo de respuesta es la unión de ambos, éste se elevará notablemente al aumentar el número de terminales.

Gráfica de distribución del tiempo de pensar

Esta gráfica se define en la cláusula 5.6.3 del estándar y representa el número de transacciones New-Order con un tiempo de pensar determinado.

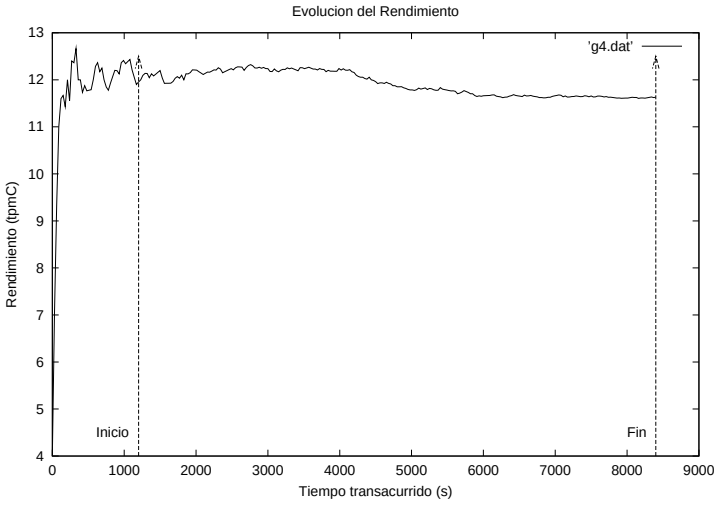


Distribución del Tiempo Pensar de Transacción New Order

Como se puede ver, la distribución aleatoria representada sigue el patrón de una distribución exponencial negativa, como se especifica en la cláusula 5.2.5.4 del estándar TPC-C. En esta gráfica también se puede ver el valor medio de la distribución.

Gráfica de evolución del rendimiento.

Esta gráfica se define en la cláusula 5.6.4 del estándar y representa la evolución del rendimiento medido a lo largo del test. El eje x representa el tiempo de test transcurrido y el eje y es el rendimiento ofrecido por el sistema hasta ese punto.



Evolución del Rendimiento

En esta gráfica se representa mediante dos líneas verticales el inicio y fin del periodo de medida. Puede apreciarse cómo el periodo de medida comienza en un estado estable tras una serie de oscilaciones iniciales en el rendimiento.

Además se puede observar cómo, aproximadamente hacia la mitad del periodo de media, el rendimiento ha disminuido levemente. Esto se debe a que postgreSQL ralentiza la ejecución de las transacciones a medida que se ejecutan operaciones sobre la base de datos. De esta circunstancia se hablará con más detalle posteriormente.

5.1.2. Test 2

Las características del sistema el el que se ha ejecutado este test con el benchmark TPCC-UVA son:

- Procesador: AMD K7 Athlon 1 Ghz.
- Memoria RAM: 1 x DIMM 256 Mb 133 Mhz.
- Memoria Caché: L1 128k, L2 512k.
- Disco Duro: Seagate Barracuda 40 Gb 7200 rpm.
- Sistema operativo: Linux Red Hat 7.2.
- kernel: 2.4.7-10.
- Compilador: gcc versión 2.96.
- Preprocesador SQL: ecpg versión 2.8.0.
- RDBMS: postgreSQL 7.1.3.

Al igual que en el test anterior no se ha utilizado ningún flag de optimización, por los motivos ya expuestos.

La prueba se realizó el día 23 de Mayo de 2002 a las 14:16:45. Los parámetros del test fueron:

- Número de almacenes: 4.
- Número de Emuladores de Terminal Remoto por almacén: 10.
- Periodo de rampa: 10 m.
- Periodo de medida: 120 m.
- Sin limpiezas en la base de datos.

Los resultados obtenidos en el sistema con la configuración descrita ha sido los siguientes:

```
RENDIMIENTO OBTENIDO: 49.000 tpmc para 4 almacenes.

 13540 Transacciones en total.
```

```
TRANSACCIONES NEW ORDER:
  5880 Transacciones en periodo de medida. 6350 Totales.
  Porcentaje: 43.427%
  Porcentaje bien hechas : 98.265%
  Tiempo de respuesta: min/med/max/90th: 0.038/1.084/11.069/2.640
  Porcentaje de transacciones rechazadas: 1.071% .
  Número medio de artículos por orden: 9.704 .
  Porcentaje de artículos remotos: 1.012% .
  Tiempo de pensar min/med/max: 0.000/12.005/108.000

TRANSACCIONES PAYMENT:
  5886 Transacciones en periodo de medida. 6354 Totales.
  Porcentaje: 43.471%
  Porcentaje bien hechas : 98.267%
  Tiempo de respuesta: min/med/max/90th: 0.008/0.936/10.014/2.480
  Porcentaje de transacciones remotas: 15.426% .
  Porcentaje de clientes seleccionados por C\_ID: 40.231% .
  Tiempo de pensar min/med/max: 0.000/12.065/120.000

TRANSACCIONES ORDER STATUS:
  591 Transacciones en periodo de medida. 640 Totales.
  Porcentaje: 4.365%
  Porcentaje bien hechas : 97.293%
  Tiempo de respuesta: min/med/max/90th: 0.025/1.273/9.372/3.040
  Porcentaje de clientes seleccionados por C\_ID: 43.316% .
  Tiempo de pensar min/med/max: 0.000/9.775/66.000

TRANSACCIONES DELIVERY:
  596 Transacciones en periodo de medida. 644 Totales.
  Porcentaje: 4.402%
  Porcentaje bien hechas : 100.000%
  Tiempo de respuesta: min/med/max/90th: 0.000/0.000/0.102/0.000
  Porcentaje ejecucion < 80s : 100.000%
  Tiempo de ejecucion min/med/max: 0.536/1.921/9.670
  N° de distritos saltados: 0 .
  Porcentaje de distritos saltados: 0.000%.
  Tiempo de pensar min/med/max: 0.000/5.148/33.000

TRANSACCIONES STOCK LEVEL:
  587 Transacciones en periodo de medida. 632 Totales.
  Porcentaje: 4.335%
  Porcentaje bien hechas : 100.000%
  Tiempo de respuesta: min/med/max/90th: 0.031/1.901/9.640/4.000
  Tiempo de pensar min/med/max: 0.000/4.952/33.000

Checkpoints con mayor duración:
Hora de comienzo           Tiempo desde el Inicio (s)  Duración (s)
Thu May 23 14:56:51 2002     2407.671000                13.955000
```

```
Thu May 23 14:26:46 2002     602.675000                  8.991000
Thu May 23 15:56:59 2002     6016.280000                 5.559000
Thu May 23 15:26:57 2002     4213.740000                 5.183000

>> TEST PASADO
```

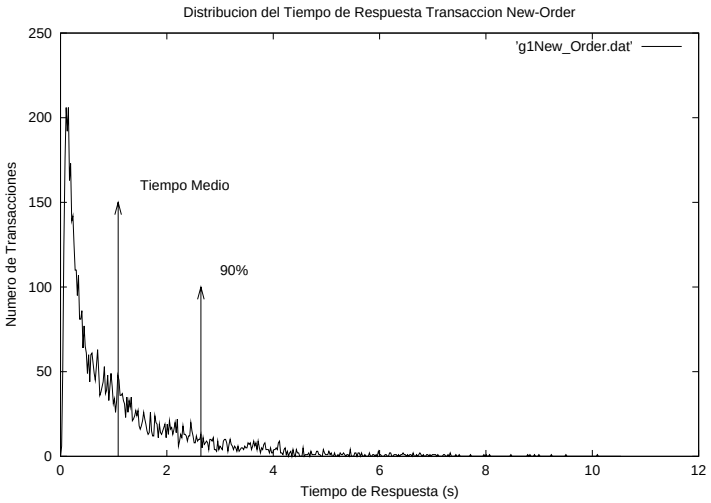
La explicación del significado de los datos obtenidos ya se hizo en el primer test, por lo que a continuación únicamente se realizará un análisis somero de los mismos.

- La tasa de rendimiento ha sido de 49 tpmC, lo que supone un rendimiento más de 4 veces superior al medido en el primer sistema.
- Se han cumplido los porcentajes de la mezcla de transacciones.
- Para las transacciones New-Order, Payment y Order-Status, los porcentajes de transacciones bien hechas y el tiempo de respuesta del 90 % se sitúan muy por debajo de los límites especificados.
- Se puede apreciar cómo para las transacciones Stock-Level y Order-Status no se han cumplido los tiempos de pensar debido a que no se han ejecutado las suficientes transacciones como para alcanzar el valor medio, al igual que en el primer test. Las variaciones de estos tiempos no superan el 5 % por lo que el impacto sobre el rendimiento es mínimo.
- En las transacciones Delivery se puede ver que ha habido un tiempo máximo de respuesta de 0.102 s. Esto se debe a que la población de terminales se ha cuadriplicado con respecto al test anterior, por lo que el sistema ha tardado más en encolar la transacción. No obstante se ve como el tiempo mínimo, medio y del 90 % sigue superando la precisión mínima utilizada por el programa.
- Los valores máximo, medio y mínimo del tiempo de ejecución de la transacción Delivery se han reducido con respecto al test anterior.
- El resto de valores cumplen las especificaciones del estándar.

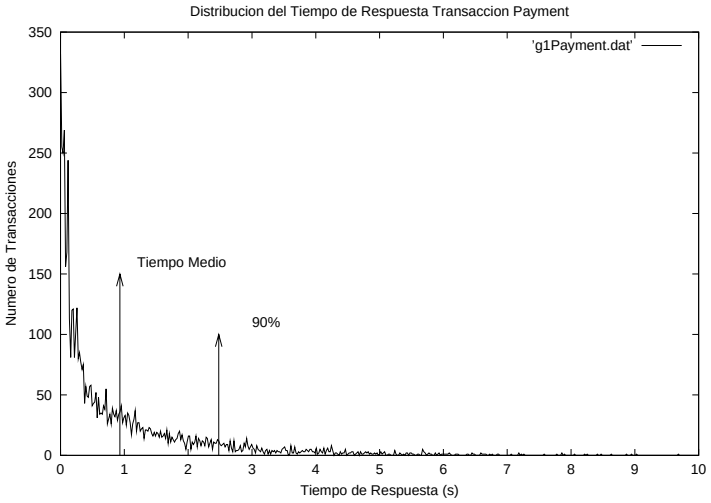
A continuación se detallan las gráficas de resultados que se han generado. La explicación de las gráficas ya se hizo en el test anterior, por lo que a continuación sólo se realizará un análisis de las mismas.

Gráficas de distribución del tiempo de respuesta

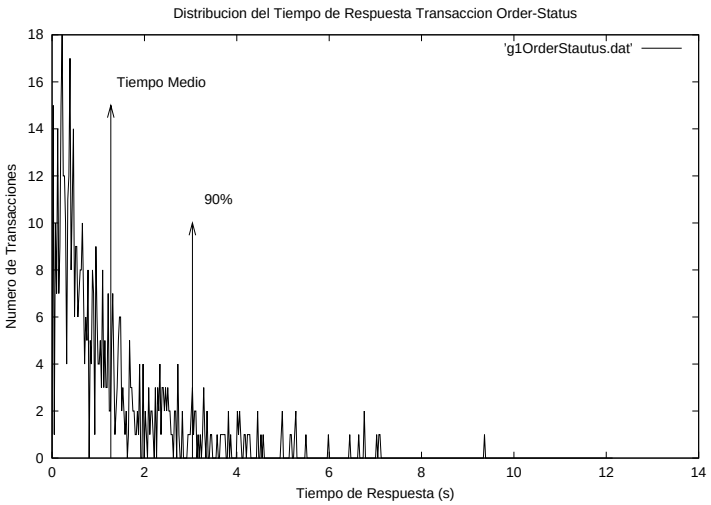
Al igual que en el test anterior no se incluye la gráfica de la transacción Delivery debido a que los tiempos de respuesta son mínimos.



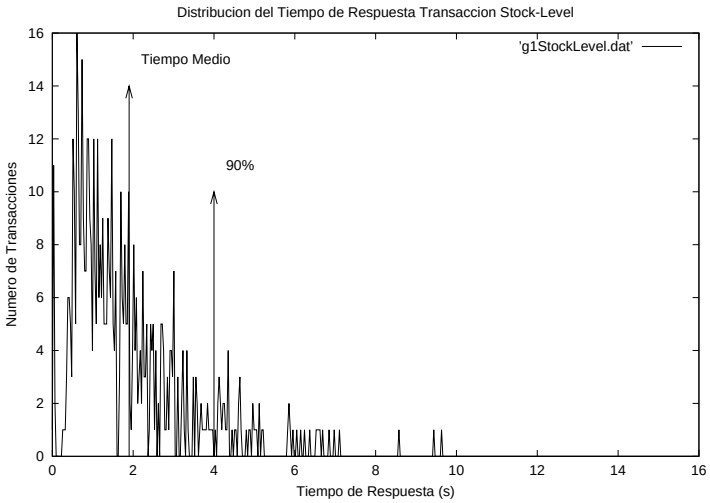
Distribución del tiempo de respuesta para la transacción New Order



Distribución del tiempo de respuesta para la transacción Payment



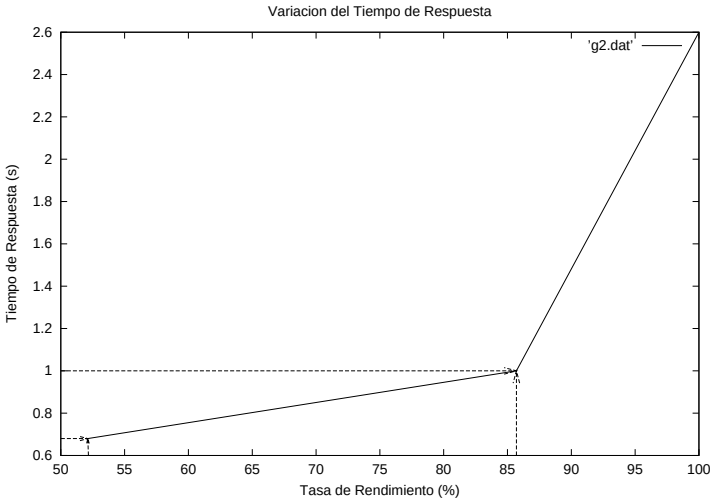
Distribución del tiempo de respuesta para la transacción Order Status



Distribución del tiempo de respuesta para la transacción Stock Level

Al igual que en el primer test, en las gráficas anteriores se puede comprobar la separación entre los tiempos medios de respuesta y el tiempo de respuesta para el 90 % de las transacciones y cómo los tiempos medios son menores que los tiempos del 90 % como requiere el estándar. También, en el caso de la Stock Level ambos tiempos son mayores que en el resto de transacciones.

Gráfica de variación del tiempo de respuesta



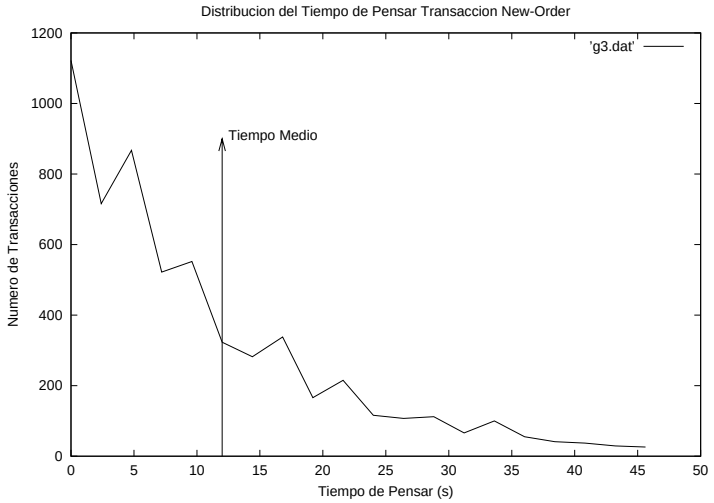
Variación del Tiempo de Respuesta de la Transacción New Order

Los puntos de esta gráfica se han obtenido de la misma forma que en el primer test.

Se puede comprobar que la disminución en el tiempo de respuesta al eliminar dos ETR sigue siendo muy acusada y que la posterior reducción de la mitad de

la población de ETRs supone un descenso menor. Por lo tanto, las conclusiones a las que se pueden llegar a partir de esta gráfica son las mismas que para el primer test.

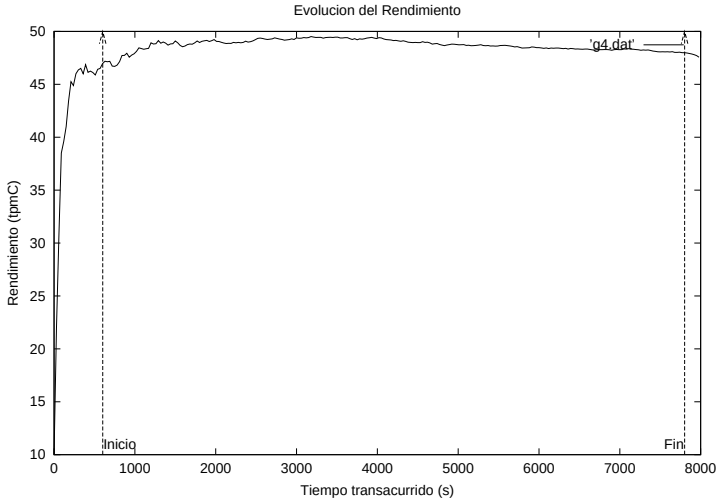
Gráfica de distribución del tiempo de pensar.



Distribución del Tiempo Pensar de Transacción New Order

Se puede comprobar que, al igual que en el primer test, la gráfica sigue una distribución exponencial negativa, cumpliendo por lo tanto con el estándar.

Gráfica de evolución del rendimiento.



Evolución del Rendimiento

En esta gráfica se puede apreciar cómo el intervalo de medida comenzó antes de que el rendimiento se estabilizara completamente, por lo que el tiempo de rampa

debería haber sido mayor. Además se observa que, una vez alcanzado el estado estable, la gráfica es más plana que en el test anterior. Esto se debe a que en este sistema postgreSQL es capaz de mantener mejor el rendimiento.

5.1.3. Test 3

Las caracteristicas del sistema en el que se ha ejecutado este test con el benchmark TPCC-UVA son:

- Procesador: AMD XP 1900+.
- Memoria RAM: 2 x DIMM 256 Mb 333 Mhz.
- Memoria Caché: L1 128k, L2 512k.
- Disco Duro: Seagate Barracuda 60 Gb 7200 rpm.
- Sistema operativo: Linux Red Hat 7.2.
- kernel: 2.4.7-10.
- Compilador: gcc versión 2.96.
- Preprocesador SQL: ecpg versión 2.8.0.
- RDBMS: postgreSQL 7.1.3.

Al igual que en primer test no se ha utilizado ningún flag de optimización, por los motivos ya expuestos.

La prueba se realizó el día 26 de Mayo de 2002 a las 22:49:54. Los parámetros del test fueron:

- Número de Almacenes: 5.
- Número de Emuladores de Terminal Remoto por almacén: 10.
- Periodo de rampa: 20 m.
- Periodo de medida: 120 m.
- Sin limpiezas en la base de datos.

Los resultados obtenidos en el sistema con la configuración descrita ha sido los siguientes:

```
RENDIMIENTO OBTENIDO: 61.375 tpmc para 5 almacenes.

 16922 Transacciones en total.
```

```
TRANSACCIONES NEW ORDER:
  7365 Transacciones en periodo de medida. 8565 Totales.
  Porcentaje: 43.523%
  Porcentaje bien hechas : 97.135%
  Tiempo de respuesta: min/med/max/90th: 0.026/1.068/18.880/2.440
  Porcentaje de transacciones rechazadas: 1.073% .
  Número medio de artículos por orden: 9.869 .
  Porcentaje de artículos remotos: 1.010% .
  Tiempo de pensar min/med/max: 0.000/12.062/108.000

TRANSACCIONES PAYMENT:
  7347 Transacciones en periodo de medida. 8564 Totales.
  Porcentaje: 43.417%
  Porcentaje bien hechas : 97.659%
  Tiempo de respuesta: min/med/max/90th: 0.008/0.874/18.733/2.200
  Porcentaje de transacciones remotas: 14.754% .
  Porcentaje de clientes seleccionados por C\_ID: 39.839% .
  Tiempo de pensar min/med/max: 0.000/12.018/120.000

TRANSACCIONES ORDER STATUS:
  735 Transacciones en periodo de medida. 860 Totales.
  Porcentaje: 4.343%
  Porcentaje bien hechas : 98.367%
  Tiempo de respuesta: min/med/max/90th: 0.013/1.004/16.044/2.360
  Porcentaje de clientes seleccionados por C\_ID: 42.041% .
  Tiempo de pensar min/med/max: 0.000/10.034/76.000

TRANSACCIONES DELIVERY:
  736 Transacciones en periodo de medida. 856 Totales.
  Porcentaje: 4.349%
  Porcentaje bien hechas : 100.000%
  Tiempo de respuesta: min/med/max/90th: 0.000/0.000/0.067/0.000
  Porcentaje ejecucion < 80s : 100.000%
  Tiempo de ejecucion min/med/max: 0.338/1.646/18.373
  N° de distritos saltados: 0 .
  Porcentaje de distritos saltados: 0.000%.
  Tiempo de pensar min/med/max: 0.000/5.247/38.000

TRANSACCIONES STOCK LEVEL:
  739 Transacciones en periodo de medida. 856 Totales.
  Porcentaje: 4.367%
  Porcentaje bien hechas : 100.000%
  Tiempo de respuesta: min/med/max/90th: 0.020/1.184/17.712/2.720
  Tiempo de pensar min/med/max: 0.000/5.181/38.000

Checkpoints con mayor duración:
Hora de comienzo           Tiempo desde el Inicio (s)   Duración (s)
Sun May 26 23:09:55 2002      1200.715000                 13.734000
```

```
Sun May 26 23:40:09 2002      3014.498000              13.690000
Mon May 27 00:40:28 2002      6633.468000              6.710000
Mon May 27 00:10:22 2002      4828.208000              5.213000

>> TEST PASADO
```

La explicación del significado de los datos obtenidos ya se hizo en el primer test, por lo que a continuación únicamente se realizará un análisis somero de los mismos.

- La tasa de rendimiento ha sido de 61,375 tpmC lo que supone un aumento frente a las configuraciones anteriores.

- Se han cumplido los porcentajes de la mezcla de transacciones.

- Al igual que en el test anterior, para las transacciones New-Order, Payment y Order-Status, los porcentajes de transacciones bien hechas y el tiempo de respuesta del 90 % se sitúan muy por debajo de los límites especificados.

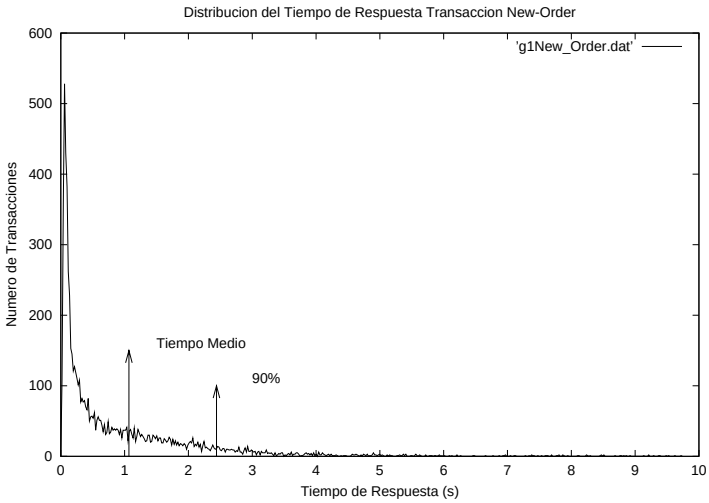
- En este test se han cumplido los tiempos medios de pensar para todas las transacciones.

- Con respecto al segundo test, los valores mínimo y medio del tiempo de ejecución de la transacción Delivery se han reducido. Por el contrario, el tiempo máximo ha aumentado. Esto puede deberse, a que alguna transacción haya encontrado la cola de mensajes del Monitor de Transacciones llena.

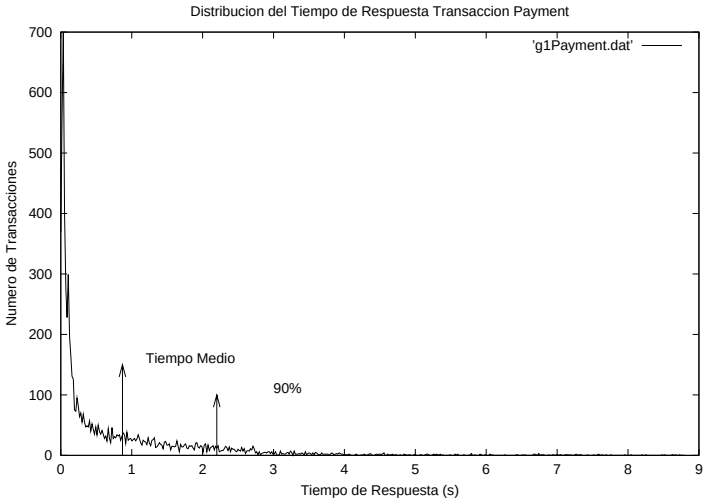
- El resto de valores cumplen las especificaciones del estándar.

A continuación se detallan las gráficas de resultados que se han generado. La explicación de las gráficas ya se hizo en el primer test, por lo que a continuación sólo se realizará un análisis somero de las mismas.

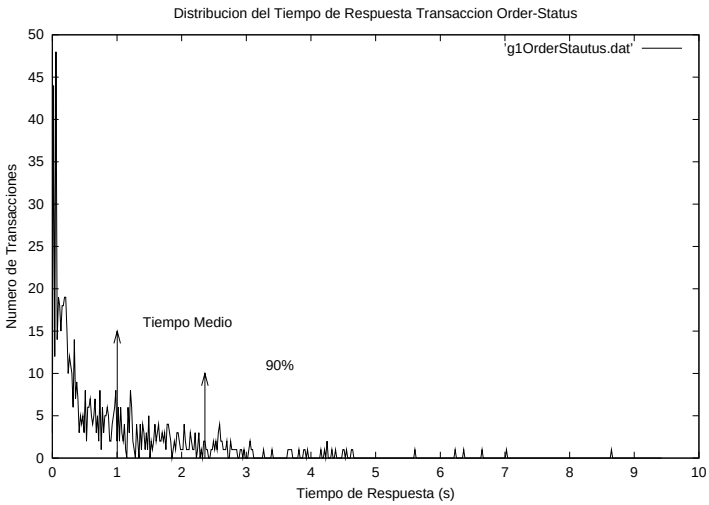
Gráficas de distribución del tiempo de respuesta



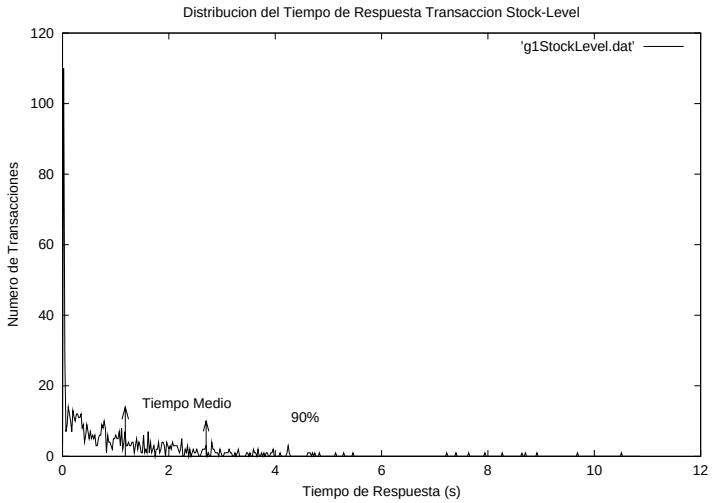
Distribución del tiempo de respuesta para la transacción New Order



Distribución del tiempo de respuesta para la transacción Payment



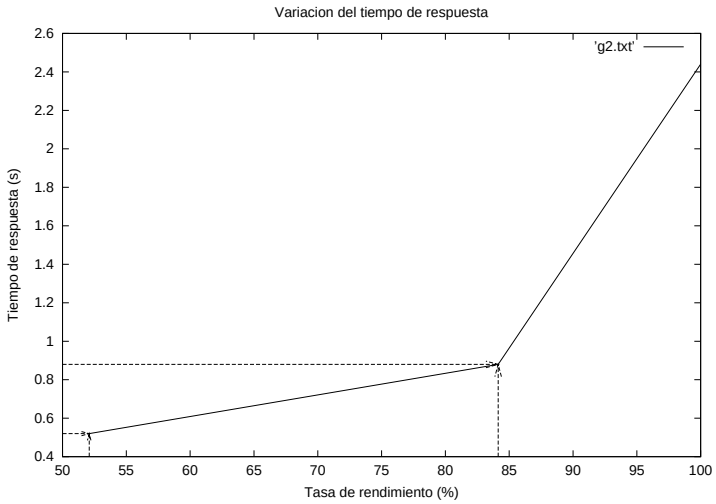
Distribución del tiempo de respuesta para la transacción Order Status



Distribución del tiempo de respuesta para la transacción Stock Level

Al igual que en el primer test, en las gráficas anteriores se puede comprobar la separación entre los tiempos medios de respuesta y el tiempo de respuesta para el 90 % de las transacciones y como los tiempos medios son menores que los tiempos del 90 % como requiere el estándar. También en el caso de la transacción Stock Level ambos tiempos son menores.

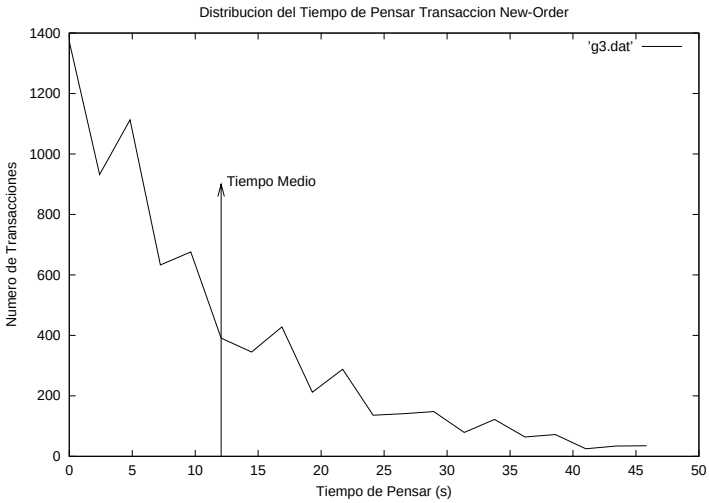
Gráfica de variación del tiempo de respuesta



Variación del Tiempo de Respuesta de la Transacción New Order

De esta gráfica se puede hacer el mismo análisis que en los tests anteriores.

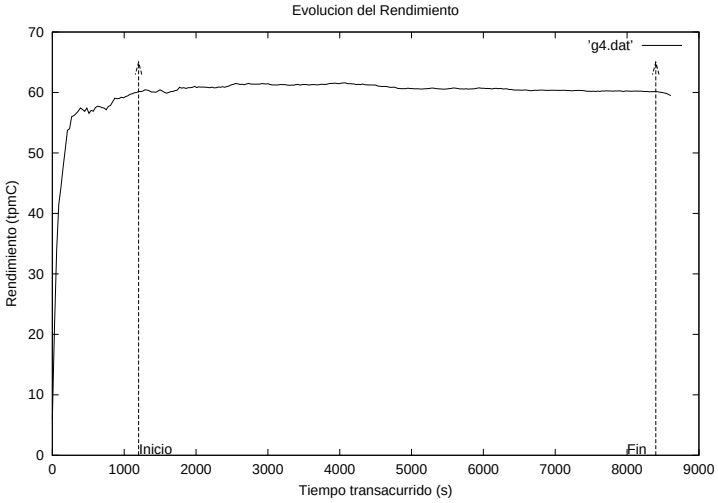
Gráfica de distribución del tiempo de pensar.



Distribución del Tiempo de Pensar de Transacción New Order

Se puede comprobar que al igual que en el primer test la gráfica sigue una distribución exponencial negativa cumpliendo, por lo tanto, con el estándar.

Gráfica de evolución del rendimiento.



Evolución del Rendimiento

A través de esta gráfica se puede observar que el periodo de medida comenzó tras alcanzar el estado estable. Además se puede comprobar que la curva es la más plana de todos los tests. Por lo tanto en este sistema es en el que postgreSQL mantiene mejor el rendimiento.

5.1.4. Análisis

Con relación a estos tres test se puede decir lo siguiente:

- Como era de esperar, se obtienen mejores resultados en sistemas más potentes, pasando de los 11,633 tpmC del primer test, a los 61,375 tpmC del tercero.
- Se ha podido comprobar en los gráficos de variación del tiempo de respuesta la influencia de la cola de mensajes en el tiempo de respuesta de las transacciones. Cuanto mayor sea la población de terminales, mayor es la probabilidad de que una transacción espere en la cola del Monitor de Transacciones. El hecho de que éste ejecute las transacciones en serie provoca que las transacciones encoladas esperen a que se ejecuten las que están por delante, aumentando con ello su tiempo de respuesta.
- Se ha podido apreciar cómo en los dos primeros test no se han cumplido los tiempos medios de pesar para algunas de las transacciones. Esto se debe a que no se han obtenido los suficientes valores como para alcanzar la media requerida. No obstante las desviaciones no superan el 5 % por lo que su impacto en el rendimiento obtenido es mínimo.

5.1.5. Tests de 8 Horas

A continuación se pasa a analizar los resultados de dos tests de 8 horas de duración, con cuatro almacenes. En el primero no se han realizado limpiezas y en el segundo se han realizado en intervalos de una hora. Los tests se han realizado sobre el mismo sistema que en el de la sección 5.1.2, que consta de:

- Procesador: AMD K7 Athlon 1 Ghz.
- Memoria RAM: 1 x DIMM 256 Mb 133 Mhz.
- Memoria Caché: L1 128k, L2 512k.
- Disco Duro: Seagate Barracuda 40 Gb 7200 rpm.
- Sistema operativo: Linux Red Hat 7.2.
- kernel: 2.4.7-10.
- Compilador: gcc versión 2.96.
- Preprocesador SQL: ecpg versión 2.8.0.
- RDBMS: postgreSQL 7.1.3.

En los siguientes resultados sólo se incluirá la gráfica de evolución de rendimiento, ya que el resto carece de relevancia para este análisis.

Test 1

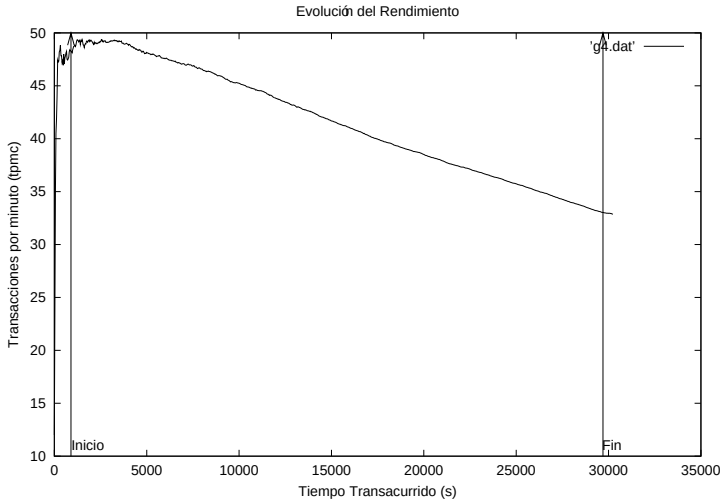
Las características del test han sido:

- Número de Almacenes: 4.
- Número de Emuladores de Terminal Remoto por almacén: 10.
- Periodo de rampa: 15 m.
- Periodo de medida: 480 m.
- Sin limpiezas en la base de datos.

Los resultados obtenidos con la configuración anterior han sido:

```
RENDIMIENTO OBTENIDO: 33.027 tpmc para 4 almacenes.

 36485 Transacciones en total.

TRANSACCIONES NEW ORDER:
  15853 Transacciones en periodo de medida. 16578 Tolales.
  Porcentaje: 43.451%
  Porcentaje bien hechas : 39.639%
  Tiempo de respuesta: min/med/max/90th: 0.040/11.703/70.569/(><)
  Porcentaje de transacciones rechazadas: 1.047% .
  Número medio de artículos por orden: 9.909 .
  Porcentaje de artículos remotos: 0.998% .
  Tiempo de pensar min/med/max: 0.000/12.112/109.000

TRANSACCIONES PAYMENT:
  15867 Transacciones en periodo de medida. 16590 Tolales.
  Porcentaje: 43.489%
  Porcentaje bien hechas : 40.518%
  Tiempo de respuesta: min/med/max/90th: 0.009/11.556/70.611/(><)
  Porcentaje de transacciones remotas: 15.252% .
  Porcentaje de clientes seleccionados por C\_ID: 40.625% .
  Tiempo de pensar min/med/max: 0.000/12.076/120.000

TRANSACCIONES ORDER STATUS:
  1588 Transacciones en periodo de medida. 1659 Tolales.
  Porcentaje: 4.352%
  Porcentaje bien hechas : 39.547%
  Tiempo de respuesta: min/med/max/90th: 0.027/11.756/64.577/(><)
  Porcentaje de clientes seleccionados por C\_ID: 43.829% .
  Tiempo de pensar min/med/max: 0.000/9.868/90.000

TRANSACCIONES DELIVERY:
  1582 Transacciones en periodo de medida. 1654 Tolales.
  Porcentaje: 4.336%
  Porcentaje bien hechas : 100.000%
  Tiempo de respuesta: min/med/max/90th: 0.000/0.000/0.009/0.000
  Porcentaje ejecucion < 80s : 100.000%
  Tiempo de ejecucion min/med/max: 0.705/12.452/70.324
  N° de distritos saltados: 0
  Porcentaje de distritos saltados: 0.000%.
  Tiempo de pensar min/med/max: 0.000/4.972/45.000

TRANSACCIONES STOCK LEVEL:
  1595 Transacciones en periodo de medida. 1658 Tolales.
  Porcentaje: 4.372%
  Porcentaje bien hechas : 72.539%
  Tiempo de respuesta: min/med/max/90th: 0.034/14.090/70.173/32.960
  Tiempo de pensar min/med/max: 0.000/4.866/45.000
```

```
Checkpoints con mayor duración:

Hora de comienzo           Tiempo desde el Inicio (s) Duración (s)
Fri May 24 23:01:03 2002    312.281000                 5.286000
Fri May 24 22:30:57 2002    507.161000                 5.083000
Sat May 25 01:01:19 2002    3528.471000                4.754000
Fri May 24 23:31:08 2002    117.821000                 4.012000

>> TEST FALLIDO
No se han cumplido los requisitos de tiempo de respuesta.
Consulte la sección 'Análisis de Resultados' del manual de usuario.
```



Evolución del Rendimiento

A partir de los resultados anteriores se pueden sacar las siguientes conclusiones:

- En la gráfica de evolución del rendimiento, se puede apreciar cómo decae a medida que avanza el test. Esto es debido a que postgreSQL almacena información residual en la base de datos a medida que se ejecutan transacciones, lo cual ralentiza su ejecución.
- Los porcentajes de las transacciones New-Order, Payment y Order-Status con tiempos de respuesta menores que 5 segundos y de Stock-Level con tiempo menor que 20 segundos, no alcanzan el 90 % requerido por el estándar TPC-C.
- En las transacciones New-Order, Payment y Order-Status, aparece el símbolo (><) en el lugar del tiempo de respuesta para el 90 % de las transacciones. Esto indica que no se ha podido calcular, puesto que es mayor que cuatro veces el valor especificado en el estándar (5 segundos).

- Los tiempos de ejecución de la transacción Delivery se han aumentado notablemente con respecto a los descritos en las secciones 5.1.1, 5.1.2, y 5.1.3.
- Comparando estos resultados con los del test de la sección 5.1.2, estos han sido peores, por lo que se puede comprobar la diferencia entre los resultados de tests de corta y larga duración.

Test 2

Las características del test han sido:

- Número de Almacenes: 4
- Número de Emuladores de Terminal Remoto por almacén: 10
- Periodo de rampa: 10 m
- Periodo de medida: 480 m
- Con limpiezas en la base de datos cada hora.

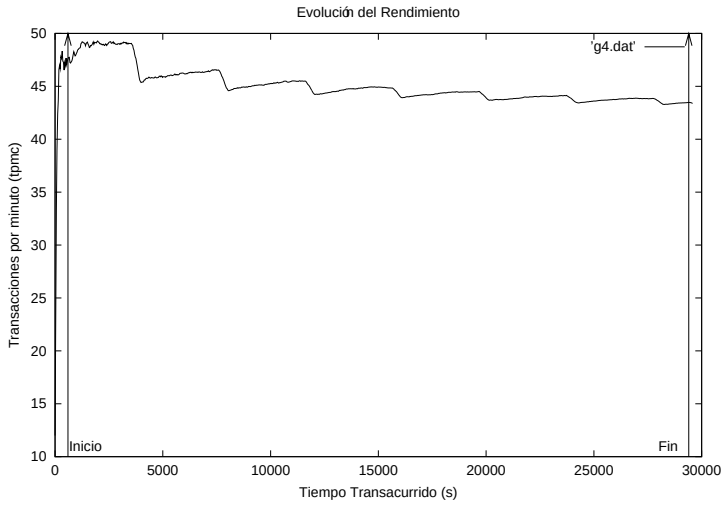
Los resultados obtenidos con la configuración anterior han sido:

```
RENDIMIENTO OBTENIDO: 43.617 tpmc para 4 almacenes.


48135 Transacciones en total.


TRANSACCIONES NEW ORDER:
20936 Transacciones en periodo de medida. 21410 Totales.
Porcentaje: 43.494%
Porcentaje bien hechas : 94.292%
Tiempo de respuesta: min/med/max/90th: 0.033/3.640/262.981/3.720
Porcentaje de transacciones rechazadas: 1.008% .
Número medio de artículos por orden: 9.924 .
Porcentaje de artículos remotos: 1.003% .
Tiempo de pensar min/med/max: 0.000/12.043/115.000


TRANSACCIONES PAYMENT:
20931 Transacciones en periodo de medida. 21403 Totales.
Porcentaje: 43.484%
Porcentaje bien hechas : 94.023%
Tiempo de respuesta: min/med/max/90th: 0.007/3.197/262.317/3.720
Porcentaje de transacciones remotas: 15.303% .
Porcentaje de clientes seleccionados por C_ID: 40.256% .
```

```
Tiempo de pensar min/med/max: 0.000/11.995/120.000


TRANSACCIONES ORDER STATUS:
2089 Transacciones en periodo de medida. 2138 Totales.
Porcentaje: 4.340%
Porcentaje bien hechas : 93.394%
Tiempo de respuesta: min/med/max/90th: 0.026/3.612/243.927/4.080
Porcentaje de clientes seleccionados por C_ID: 43.226% .
Tiempo de pensar min/med/max: 0.000/9.875/90.000


TRANSACCIONES DELIVERY:
2087 Transacciones en periodo de medida. 2137 Totales.
Porcentaje: 4.336%
Porcentaje bien hechas : 100.000%
Tiempo de respuesta: min/med/max/90th: 0.000/0.000/0.193/0.000
Porcentaje ejecucion <80s : 99.138%
Tiempo de ejecucion min/med/max: 0.537/4.469/242.659
Nº de distritos saltados: 0 .
Porcentaje de distritos saltados: 0.000%.
Tiempo de pensar min/med/max: 0.000/4.978/45.000


TRANSACCIONES STOCK LEVEL:
2092 Transacciones en periodo de medida. 2137 Totales.
Porcentaje: 4.346%
Porcentaje bien hechas : 97.610%
Tiempo de respuesta: min/med/max/90th: 0.022/4.441/237.119/4.640
Tiempo de pensar min/med/max: 0.000/4.851/45.000


Checkpoints con mayor duración:
 Hora de comienzo          Tiempo desde el Inicio (s)  Duración (s)
 Sat May 25 15:00:08 2002         7814.349000           22.720000
 Sat May 25 19:31:05 2002        24071.936000           11.895000
 Sat May 25 15:30:30 2002         9637.112000            5.301000
 Sat May 25 20:31:20 2002        27686.947000            5.210000

Vacuums con mayor duración:
 Hora de comienzo          Tiempo desde el Inicio (s)  Duración (s)
 Sat May 25 18:18:16 2002        19702.855000          451.607000
 Sat May 25 19:25:48 2002        23754.511000          450.266000
 Sat May 25 16:04:00 2002        11646.438000          429.448000
 Sat May 25 17:11:09 2002        15675.919000          426.883000



>> TEST PASADO
```

En los resultados anteriores aparece información acerca de las cuatro limpiezas (vacuums) de mayor duración. Consta de la fecha y hora del comienzo de cada limpieza, del tiempo de test en segundos transcurrido al inicio de la limpieza y la duración en segundos de ésta.

Se puede apreciar como la duración de las Limpiezas ronda los 7.5 segundos.



Evolución del Rendimiento

A partir de los resultados anteriores se pueden sacar las siguientes conclusiones:

- En la gráfica de evolución del rendimiento se puede comprobar cómo a partir de la segunda limpieza, éste ha suavizado su caída con respecto al test anterior. También, se puede apreciar cómo se produce una perturbación en la curva de rendimiento en el instante en el que se ejecuta la limpieza.
- El rendimiento ha aumentado un 24 % con respecto al test anterior, pasando de 33,027 tpmC a 43,617 tpmC.
- A diferencia del test anterior, se han cumplido los porcentajes de transacciones ejecutadas en un tiempo menor que el especificado por el TPC-C.
- Los tiempos de respuesta para el 90 % de las transacciones están por debajo de los valores especificados.
- El tiempo medio de respuesta se acerca al valor del tiempo de respuesta del 90 % de las transacciones. Esto es debido al bloqueo de la tablas de la base de datos producido durante la ejecución de la limpieza, que provoca que las transacciones que se ejecutan en paralelo con ésta tengan un tiempo de respuesta mayor. El aumento del tiempo de respuesta de estas transacciones provoca el aumento del tiempo medio, pero por contra afecta mínimamente al tiempo de respuesta del 90 % de las transacciones, ya que son pocas las que se ejecutan durante la limpieza.

- Los tiempos de respuesta de las transacciones New-Order, Payment, Stock-Level, y Order-Status, y el tiempo de ejecución de la transacción Delivery, tienen todos valores máximos rondando los 4.5 minutos. Estos tiempos son los de las transacciones que se han ejecutado en paralelo con alguna de las limpiezas efectuadas en la base de datos, ya que los tiempos medios son mucho menores.

- Los resultados del test con limpiezas han mejorado con respecto al test anterior, pasando de 33,027 tpmC, a 43,617 tpmC. Sin embargo se puede comprobar que en el test de dos horas descrito en el apartado 5.1.2 se han obtenido mejores resultados (49,0 tpmC) que en este (43,617 tpmC). Como conclusión la ejecución de limpiezas durante el test suaviza la caída del rendimiento en tests de larga duración, no obstante, se sigue notando la disminución de rendimiento con respecto a tests de menor duración.

5.2. Limitaciones encontradas

A la vista de los resultados se pueden sacar las siguientes conclusiones acerca de las limitaciones de esta implementación del benchmark:

- El Monitor de Transacciones procura el aislamiento entre las transacciones ejecutándolas en serie según su orden de llegada. Esto supone tener transacciones en espera, siendo frecuente que los tiempos de respuesta superen los especificados. Cuando una transacción espera en la cola, su tiempo de respuesta es la suma de su tiempo de ejecución más la suma de los tiempos de ejecución de las transacciones por delante en la cola. Sería necesario desarrollar un Monitor de Transacciones que redujera los tiempo de repuesta paralelizando la ejecución del las transacciones y procurando el aislamiento de las mismas.

  A pesar de esta limitación, la información presentada sigue siendo relevante. El hecho de que en una maquina no se hayan cumplido los tiempos de respuesta, no impide al usuario tomar decisiones acerca de la eficiencia de una máquina.

- Debido a que postgreSQL no es capaz de mantener el rendimiento durante periodos de larga duración, es necesaria la ejecución periódica de limpiezas. Esto afecta, como se puede ver en las gráficas, a la evolución del rendimiento. No obstante, se puede apreciar que a partir de un punto la evolución del rendimiento se estabiliza. Este efecto de postgreSQL reduce la posibilidad de encontrar un test cuya variación del rendimiento cumpla con el estándar, es decir, que se pueda mantener el mismo rendimiento tanto en pruebas de 2 horas como en pruebas de 8 horas. Como conclusión, para realizar comparativas entre distintas máquinas se debe buscar una configuración para la cual la evolución del rendimiento durante el periodo de medida sea estable, y comparar únicamente los resultados obtenidos durante el periodo de medida, que deberá ser el mismo para ambas pruebas.

- Debido a las limitaciones anteriores, los resultados obtenidos con TPCC-UVA no son comparables con los obtenidos con otras implementaciones del TPC-C. Se debe tener en cuenta, además, que el Monitor de Transacciones que se utiliza está diseñado a medida para esta implementación por lo que no es un monitor comercial como requiere el estándar. Por otra parte, el uso de un monitor de transacciones comercial supondría que el TPCC-UVA no se podría distribuir bajo licencia pública. En conclusión, los resultados obtenidos a través de la ejecución de TPCC-UVA sólo son válidos al objeto de comparar el rendimiento de diferentes sistemas en los que se ha ejecutado este software.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones.

Tras la finalización del proyecto se pueden sacar las siguientes conclusiones:

- Se ha implementado el Benchmark TPC-C del *Transaction Processing Performance Council* (TPC) para medir el rendimiento de sistemas distribuidos de procesamiento de transacciones en línea (OLTP).
- Se ha realizado la una implementación de dominio público del estándar TPC-C, lo que viene a cubrir un vacío importante en el área de los benchmark de libre distribución.
- Se ha diseñado un benchmark susceptible de ser utilizado en cualquier entorno UNIX/Linux compatible con el estándar System V.
- Se ha comunicado cada uno de los procesos que intervienen en la medición de rendimientos a través de mecanismos IPC (Inter Process Communication). Gracias a esto, el benchmark podrá utilizarse en el futuro para comprobar la eficiencia de un cluster basado en memoria compartida distribuida, como el que se está desarrollando en el Area de Arquitectura del Ordenadores del Departamento de Informática de la Universidad de Valladolid.
- Se han cumplido las especificaciones del TPC-C referentes al uso de gestores de base de datos disponibles en el mercado, utilizando el motor postgreSQL de libre distribución.
- Se ha implementado un esquema de base de datos que cumple con los requisitos del estándar.
- Se ha diseñado un Emulador de Terminal Remoto cumpliendo con los requisitos del TPC-C.

- Se ha diseñado un Monitor de Transacciones que permite la ejecución de transacciones con el nivel de aislamiento requerido por el estándar.
- Se ha diseñado un módulo de control que sincroniza los procesos que intervienen en el test y permite al usuario elegir las operaciones a realizar.
- Se ha conseguido medir el rendimiento de máquinas uniprocesador, pudiendo atestiguar el funcionamiento del benchmark.

6.2. Trabajo futuro

Algunas líneas de trabajo que se pueden seguir si se desea en un futuro seguir desarrollando esta implementación del benchmark son:

Para mejorar la ejecución de las transacciones:

- Mejorar el Monitor de Transacciones actual, con el fin de que sea posible la ejecución de transacciones en paralelo, manteniendo el aislamiento entre ellas.
- Otra solución es la utilización de un Monitor de Transacciones comercial, cumpliendo con las especificaciones del estándar.

Para mejorar el rendimiento del motor de base de datos:

- Optimizar la configuración de postgreSQL para conseguir una mejor eficiencia en la ejecución de las transacciones, aumentando a la vez el rendimiento del sistema.
- Otra solución es la utilización de otros motores de base de datos comerciales que ofrezcan mejores características que postgreSQL, en lo que a velocidad de ejecución se refiere.

Para mejorar las características del terminal:

- Mejorar el interfaz con el usuario del Emulador de Terminal Remoto.

Para comprobar la portabilidad del benchmark:

- Probar el benchmark en otros entornos UNIX y en otras arquitecturas distintas a los PC's, para comprobar el funcionamiento de los mecanismos IPC que se han utilizado en las comunicaciones entre el Monitor de Transacciones y los Emuladores de Terminal Remoto.

Apéndice A

Especificaciónes del benchmark de transacciones distribuidas TPC-C

En este apéndice se describen en detalle las cláusulas de que se compone la versión 5 del estándar del Benchmark de Transacciones Distribuidas TPC-C [6]. En las especificaciones se detallan todos los requisitos que debe cumplir el software para que se le pueda considerar un benchmark TPC-C, además de las consideraciones que ha tener en cuenta en lo que respecta a la medición de rendimientos.

En esta descripción no se han incluido las cláusulas 7, 8 y 9 del estándar ya que su contenido carece de relevancia para la explicación del benchmark implementado.

En la descripción se ha intentado respetar al máximo el documento original, con el fin de no distorsionar su contenido.

Asímismo, la numeración de las cláusulas y de sus subapartados se coresponde a la del documento original, para facilitar al lector su consulta si así lo desea. En el CD adjunto se incluye el documento original de las especificaciones del TPC-C.

Cláusula 0: Preámbulo

0.1 Introducción

El benchmark TPC-C simula las operaciones de acceso y actualización de datos realizadas en los entornos de trabajo OLTP (Online Transaction Protocol) poniendo en juego un amplio conjunto de componentes asociados a dicho entorno que se caracterizan por:

- Ejecutar simultáneamente múltiples tipos de transacciones de complejidad diferente.
- Tener modos de ejecución de transacciones en línea y aplazado.
- Soportar múltiples sesiones de terminal en línea.
- Presentar tiempos de sistema y ejecución de aplicación moderados.
- Procurar un número significativo de lecturas y escrituras en el disco.
- Cumplir la integridad de las transacciones (Reglas ACID).
- El acceso con distribución no uniforme a los datos a través de las llaves primaria y secundaria.
- Una base de datos compuesta de tablas con una amplia variedad de tamaños, atributos y relaciones.
- El compromiso entre el acceso y la actualización de los datos.

El TPC-C mide el número de ordenes procesadas por minuto en un sistema OLTP. En esta medida, se usan múltiples tipos de transacciones, que están sujetas a un tiempo de respuesta mínimo, para simular la operación de procesar una orden comercial. La unidad de medida de este benchmark son las *transacciones por minuto C* (tpmC). Para ajustarse al estándar TPC-C, todos los resultados deben incluir la tasa tpmC, el precio asociado por tpmC y la fecha de disponibilidad de la configuración propuesta.

Aunque estas especificaciones explican la implementación a través de un modelo relacional de datos con un esquema convencional de bloqueo, la base de datos puede implementarse usando cualquier motor de base de datos (DBMS), servidor de base de datos, sistema de ficheros u otro contenedor de datos que ofrezca una implementación con funcionamiento equivalente. Los términos *tabla*, *fila* y *columna* se usan en este documento solamente como ejemplos de estructuras lógicas de datos.

El TPC-C usa terminología y unidades de medida similares a otros benchmarks, ya sean provenientes de TPC u otros. Dicha similitud en terminología no implica que los resultados del TPC-C sean comparables con otros benchmarks. Los únicos resultados comparables con el TPC-C son aquellos obtenidos en un TPC- C que se ajuste a la misma revisión.

A pesar de que este benchmark ofrece un entorno que emula muchas aplicaciones OLTP, no incluye la totalidad de las operaciones realizadas en dicho entorno, por lo que el rendimiento de una aplicación de un cliente se aproximará más a los resultados declarados, cuanto más se asemeje al benchmark. El rendimiento relativo de los sistemas proporcionado por este benchmark no se mantiene necesariamente para otros espacios de trabajo o entornos. No es recomendable extrapolar los resultados a otros entornos.

Los resultados del benchmark dependen directamente de la carga de trabajo, de los requisitos específicos de la aplicación y del diseño del sistema y la implementación. El rendimiento relativo del sistema variará como resultado de esos y de otros factores. Por lo tanto, el TPC-C no debe sustituir a un benchmark especifico para una aplicación de cliente en la que se contemplen decisiones críticas en cuanto a las capacidades de planificación y evaluación del producto.

Se permite a los patrocinadores del benchmark diseñar multitud de posibles sistemas, siempre y cuando cumplan con el modelo descrito e ilustrado gráficamente en la cláusula 6. Se deberá ofrecer, junto con los resultados, un informe completo sobre los detalles de la implementación, como se especifica en la cláusula 8.

0.2 Pautas Generales de Implementación

El propósito de los benchmarks TPC es ofrecer a los usuarios industriales un resultado sobre el rendimiento relevante y objetivo. Para alcanzar este propósito, las especificaciones del benchmark requieren que los test benchmark sean implementados en sistemas, productos, tecnologías y precios que:

- Estén disponibles, de forma general, a los usuarios.
- Estén relacionados con el segmento de mercado que el benchmark TPC represente (ej: TPC-A modela y representa un entorno OLTP único, de gran volumen).
- Implemente un número significativo de usuarios en el segmento de mercado que el benchmark modela o representa.

El uso de nuevos sistemas, productos, tecnologías (hardware o software) y precios está recomendado siempre y cuando se cumplan con los requisitos mencionados previamente. Están específicamente prohibidos los sistemas de benchmark, productos, tecnologías y precios destinados únicamente al la optimización de los resultados del benchmark TPC sin que se correspondan con las aplicaciones y los entornos reales. En otras palabras se prohiben todas las implementaciones benchmark *especiales* que mejoren los resultados de benchmark pero no el rendimiento ni el precio real.

Se deben emplear las siguientes características para juzgar si una implementación particular es un benchmark *especial*. No se requiere el cumplimiento de cada uno de los puntos, pero se considera el sentido común para identificar una implementación no válida.

Se tendrán en cuenta las siguientes cuestiones para juzgar si una implementación particular es un benchmark especial:

- ¿La aplicación esta disponible, de forma general, documentada y respaldada?
- ¿La implementación tiene restricciones significativas que limitan su uso más allá de los benchmarks TPC?
- ¿La implementación o parte de la implementación está pobremente integrada dentro de un producto mayor?
- ¿La implementación toma una especial ventaja de la naturaleza limitada de los benchmark TPC (ej: perfil de transacción, mezcla de transacciones, concurrencia o compromiso entre transacciones, aislamiento de transacciones) de manera que no sería aplicable en general en el entorno que el benchmark representa?
- ¿El uso de la implementación está desaconsejada por el vendedor? (Esto incluye evitar promocionar la implementación de una forma similar a otros productos y tecnologías.)
- ¿Requiere la implementación una sofisticación poco común por parte del usuario final, programador, o administrador del sistema?
- ¿Es el precio poco habitual por parte del vendedor o poco habitual en lo referente a las practicas comerciales normales? Se consideran sospechosas las siguientes prácticas:
  - Disponibilidad de un descuento a un pequeño grupo de posibles clientes.
  - Descuentos documentados de una forma poco habitual.
  - Descuentos que excedan del 25 % en pequeñas cantidades y del 50 % en grandes cantidades.
  - Restricciones inusuales o poco frecuentes en la transferencia del producto, garantía o mantenimiento en artículos con descuento.
- ¿Los usuarios finales usan o compran la implementación (incluyendo la beta) en el segmento de mercado que el benchmark representa? ¿Cuántos? ¿En varios sitios? Si la implementación no está usándose actualmente, ¿hay alguna evidencia que indique que será usada por un número significativo de usuarios?

0.3 Pautas Generales en la Medida.

Se espera que los resultados del benchmark TPC sean representaciones precisas del rendimiento del sistema. Por lo tanto, hay ciertas pautas que se deben seguir cuando se midan esos resultados. El planteamiento o la metodología está descrita explícitamente en la especificación.

- El planteamiento es una practica o estándar aceptado en ingeniería.
- El planteamiento no realza el resultado.
- El equipo usado en la medida de los resultados está calibrado de acuerdo a los estándares de calidad establecidos.

- Se mantienen la fidelidad y la franqueza para declarar cualquier anomalía en los resultados, incluso si está especificado en los requisitos del benchmark.

Se recomienda el uso de metodologías y procedimientos nuevos siempre y cuando se ajusten a los requerimientos mencionados arriba.

Cláusula 1: Diseño Lógico de la Base de Datos

1.1 - Entornos de negocio y aplicación

El benchmark TPC-C está compuesto de un conjunto de operaciones básicas diseñadas para que los componentes del sistema representen la actividad los entornos complejos de aplicación *OLTP*. Esas operaciones básicas se han dado bajo un contexto real, describiendo la actividad de un proveedor de productos al por mayor, para ayudar al usuario a relacionarse con los componentes del benchmark de una forma intuitiva. La carga de trabajo se basa en el procesamiento de órdenes y ofrece un diseño lógico de base de datos que puede ser distribuido sin cambios en la estructura de las transacciones. El TPC-C no representa la actividad de ningún segmento comercial en particular, sino más bien cualquier tipo de industria que deba gestionar, vender o distribuir productos o servicios (ej: alquiler de coches, distribución de comida, proveedores de piezas, etc.) El TPC-C no tiene como objetivo ser un modelo de cómo construir una aplicación real.

El propósito de un benchmark es reducir la diversidad de operaciones que se encuentran en una aplicación de producción, conservando mientras tanto las características esenciales de funcionamiento, como son el nivel de utilización del sistema y la complejidad de las operaciones.

Para gestionar un sistema de entrada de órdenes de producción, tienen que realizarse un amplio número de operaciones. Muchas de esas operaciones no son de principal interés en el análisis de rendimiento, puesto que son relativamente *cortas* en términos de utilización de los recursos del sistema o en términos de frecuencia de ejecución. A pesar de que esas operaciones son vitales en un sistema productivo, crean excesiva diversidad en el contexto de un estándar de benchmark y han sido omitidas en el TPC-C.

La compañía descrita por el benchmark es un proveedor de productos al por mayor que posee un número de distritos de venta distribuidos geográficamente y que están relacionados con diversos almacenes. A medida que la estructura de la compañía aumenta, se crean nuevos almacenes con distritos asociados. Cada almacén local cubre 10 distritos. Cada distrito sirve a 3.000 clientes. Todos los almacenes mantienen existencias de los 100.000 artículos que vende la compañía. En la figura A.1 se ilustra la jerarquía de almacén, distrito y cliente del entorno de negocios del TPC-C.

Los clientes pueden mandar una nueva orden a la compañía o consultar el estado de una orden anterior. Las ordenes están compuestas por un número medio de diez líneas de orden (o artículos). Un uno por ciento de todas las líneas de orden corresponden a artículos que no pertenecen a las existencias del almacén local y deben ser suministradas por otro almacén.

Se usa, además, el sistema de la compañía para registrar pagos de clientes, procesar ordenes de entrega, y comprobar los niveles de la existencias para identificar posibles fallos en el suministro.

1.2 Entidades de la base de datos, Relaciones y Características

1.2.1 La base de datos del TPC-C se compone de nueve tablas individuales y separa-

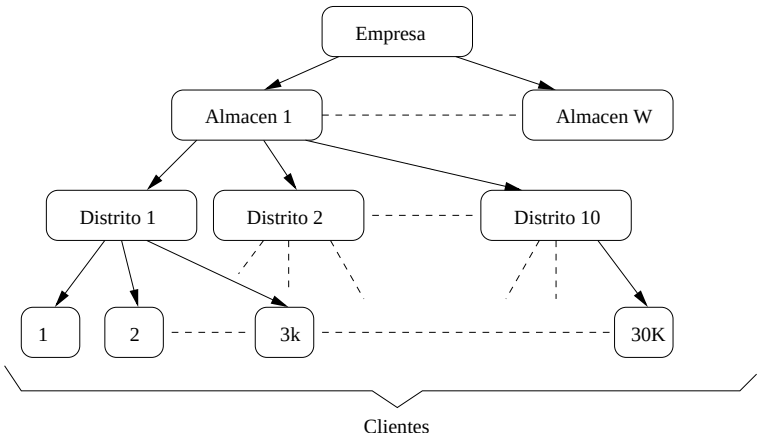


Figura A.1: Esquema de la compañía

das. Las relaciones entre esas tablas se definen en el diagrama de entidad-relación de la figura A.2 y está sujeto a las reglas que se especifican en la Cláusula 1.4.

1.3 Formato de las Tablas

1.3.1 La siguiente lista define la estructura mínima (lista de atributos) de cada tabla, donde:

- N Ids. único: significa que el atributo debe ser capaz de almacenar cualquier ID dentro de un conjunto mínimo de N IDs. únicas, sin tener en cuenta la representación física (ej. binaria, decimal, alfabética, etc) del atributo.
- Texto variable, tamaño N: significa que el atributo debe ser capaz de mantener cualquier cadena de caracteres de longitud variable con una longitud máxima de N. Si el atributo se almacena como una cadena de longitud fija y la cadena que contiene es de menos de N caracteres, debe ser rellenada con espacios.
- Texto fijo, tamaño N: significa que el atributo es capaz de almacenar cualquier cadena de caracteres con una longitud fija de N.
- Fecha y Hora: significa que el atributo debe ser capaz de contener cualquier fecha entre el 1 de Enero de 1900 y el 31 de Diciembre de 2100 con una resolución de al menos 1 segundo.
- Numérico, N dígitos: significa que el atributo debes ser capaz de contener un valor de N dígitos decimales. Los campos numéricos que contienen valores de moneda (W_YTD, S_YTD, C_CREDIT_LIM, C_BALANCE, C_YTD_PAYMENT, H_AMOUNT, OL_AMOUNT, I_PRICE) deben usar tipos de datos que ofrezcan una representación exacta de al menos la unidad monetaria más pequeña de la moneda usada en el benchmark implementado. Por ejemplo, C_BALANCE en dólares de EE.UU debe representarse como un decimal con signo de (12,2) dígitos (con escalado implícito), o escalado en centavos en un entero con signo de al menos 41 bits, o escalado en céntimos con un numero real de precisión doble (64 bits).

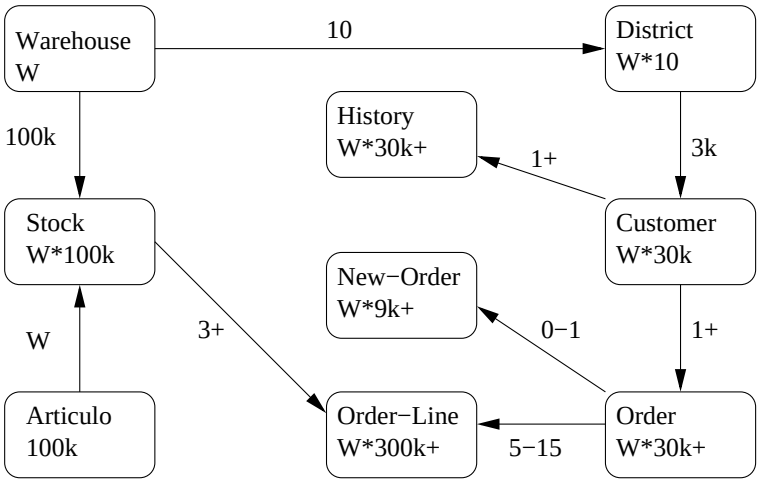


Figura A.2: Esquema de la base de datos

Leyenda:

- Todos los números señalados ilustran los requisitos de población de la base de datos.
- Los números en los bloques de entidad representan la cardinalidad de las tablas (número de filas). Esos números tienen como factor a W, el número de almacenes, a fin de ilustrar el escalado de la base de datos (ver cláusula 4).
- Los números próximos a las flechas de interrelación representan la cardinalidad de las relaciones (numero medio de hijos por padre).
- Se usa el signo más (+) detrás de la cardinalidad de la relación o de la tabla para indicar que ese número está sujeto a pequeñas variaciones sobre el escalado inicial de la base de datos durante el intervalo de medida (ver cláusula 5.5) a medida que se insertan o eliminan filas.

- 'Null': significa fuera del rango de valores válidos por un atributo y siempre el mismo valor para dicho atributo.

Comentario: Los nombres de la tabla y de los atributos tienen un propósito explicativo, la implementación puede usar nombres diferentes.

Formato de la Tabla WAREHOUSE:

Nombre de Campo	Definición del campo	Comentarios
W_ID	2*W Ids únicas	Se pueblan W almacenes
W_NAME	texto variable, tamaño 10	
W_STREET_1	texto variable, tamaño 20	
W_STREET_2	texto variable, tamaño 20	
W_CITY	texto variable, tamaño 20	
W_STATE	texto fijo, tamaño 2	
W_ZIP	texto fijo, tamaño 9	
W_TAX	numérico, 4 dígitos	Tasa de ventas
W_YTD	Numérico, 12 dígitos	Balance de año hasta la fecha

Clave Primaria: W_ID

Formato de la Tabla DISTRICT:

Nombre de Campo	Definición del campo	Comentarios
D_ID	20 Ids únicos	Se pueblan 10 por almacén
D_W_ID	2*W Ids únicos	
D_NAME	texto variable, tamaño 10	
D_STREET_1	texto variable, tamaño 20	
D_STREET_2	texto variable, tamaño 20	
D_CITY	texto variable, tamaño 20	
D_STATE	texto fijo, tamaño 2	
D_ZIP	texto fijo, tamaño 9	
D_TAX	Numérico, 4 dígitos	Tasa de venta
D_YTD	Numérico, 12 dígitos	Balance de Año hasta la fecha
D_NEXT_O_ID	10,000,000 Ids únicos	Siguiente número de orden disponible

Clave primaria: (D_W_ID, D_ID)

(D_W_ID) Clave foránea, referencia a (W_ID)

Formato de la Tabla CUSTOMER:

Nombre de Campo	Definición del campo	Comentarios
C_ID	96,000 Ids únicos	Se pueblan 3,000 por distrito
C_D_ID	20 Ids únicos	
C_W_ID	2*W Ids únicos	
C_FIRST	texto variable, tamaño 16	
C_MIDDLE	texto fijo, tamaño 2	
C_LAST	texto variable, tamaño 16	
C_STREET_1	texto variable, tamaño 20	
C_STREET_2	texto variable, tamaño 20	
C_CITY	texto variable, tamaño 20	
C_STATE	texto fijo, tamaño 2	
C_ZIP	texto fijo, tamaño 9	
C_PHONE	texto fijo, tamaño 16	
C_SINCE	Fecha y hora	
C_CREDIT	Texto fijo, tamaño 2	'GC'=bueno, 'BC'=malo
C_CREDLIM	numérico, 12 dígitos	
C_DISCOUNT	numérico, 4 dígitos	
C_BALANCE	Numérico con signo, 12 dígitos	
C_YTD_PAYMENT	Numérico, 12 dígitos	
C_PAYMENT_CNT	Numérico, 4 dígitos	
C_DELIVERY_CNT	Numérico, 4 dígitos	
C_DATA	Texto variable, tamaño 500	Información Miscelánea

Clave primaria: (C_W_ID, C_D_ID, C_ID)

(C_W_ID, C_D_ID) Clave foránea, referencia a (D_W_ID, D_ID)

Formato de la Tabla HISTORY:

Nombre de Campo	Definición del campo	Comentarios
H_C_ID	96,000 Ids únicos	
H_C_D_ID	20 Ids únicos	
H_C_W_ID	2*W Ids únicos	
H_D_ID	20 Ids únicos	
H_W_ID	2*W Ids únicos	
H_DATE	fecha y hora	
H_AMOUNT	numérico, 6 dígitos	
H_DATA	texto variable, tamaño 24	Información Miscelánea

Clave primaria: ninguna

(H_C_W_ID, H_C_D_ID, H_C_ID) Clave foránea, referencia a (C_W_ID, C_D_ID, C_ID)

(H_W_ID, H_D_ID) Clave foránea, referencia a (D_W_ID, D_ID)

Comentario: Las filas de la tabla History no tienen llave primaria ya que en el contexto del benchmark no se necesita identificar unívocamente sus filas.

Nota: La aplicación TPC-C no tiene que ser capaz de utilizar el rango incrementado de los valores C_ID más allá de 6,000.

Formato de la Tabla NEW-ORDER:

Nombre de Campo	Definición del campo	Comentarios
NO_O_ID	10,000,000 Ids únicos	
NO_D_ID	20 Ids únicos	
NO_W_ID	2*W Ids únicos	

Clave primaria: (NO_W_ID, NO_D_ID, NO_O_ID)

(NO_W_ID, NO_D_ID, NO_O_ID) Clave foránea, referencia (O_W_ID, O_D_ID, O_ID)

Formato de la Tabla ORDER

Nombre de Campo	Definición del campo	Comentarios
O_ID	10,000,000 Ids únicos	
O_D_ID	20 Ids únicos	
O_W_ID	2*W Ids únicos	
O_C_ID	96,000 Ids únicos	
O_ENTRY_D	fecha y hora	
O_CARRIER_ID	10 Ids únicos , o Null	
O_OL_CNT	de 5 a 15	Contador de Order-Lines
O_ALL_LOCAL	numérico, 1 dígito	

Clave primaria: (O_W_ID, O_D_ID, O_ID)

(O_W_ID, O_D_ID, O_C_ID) Clave foránea, referencia (C_W_ID, C_D_ID, C_ID)

Formato de la Tabla ORDER-LINE

Nombre de Campo	Definición del campo	Comentarios
OL_O_ID	10,000,000 Ids únicos	
OL_D_ID	20 Ids únicos	
OL_W_ID	2*W Ids únicos	
OL_NUMBER	15 Ids únicos	
OL_I_ID	200,000 Ids únicos	
OL_SUPPLY_W_ID	2*W Ids únicos	
OL_DELIVERY_D	fecha y hora, o Null	
OL_QUANTITY	numérico, 2 dígitos	
OL_AMOUNT	numérico, 6 dígitos	
OL_DIST_INFO	texto fijo, tamaño 24	

Clave primaria: (OL_W_ID, OL_D_ID, OL_O_ID, OL_NUMBER)

(OL_W_ID, OL_D_ID, OL_O_ID) Clave foránea, referencia (O_W_ID, O_D_ID, O_ID)

(OL_SUPPLY_W_ID, OL_I_ID) Clave foránea, referencia (S_W_ID, S_I_ID)

Formato de la Tabla ITEM:

Nombre de Campo	Definición del campo	Comentarios
I_ID	200,000 Ids únicos	Se pueblan 100,000 artículos
I_IM_ID	200,000 Ids únicos	ID de Imagen asociada al artículo
I_NAME	texto variable, tamaño 24	
I_PRICE	numérico, 5 dígitos	
I_DATA	texto variable, tamaño 50	Información de la marca

Clave primaria: I_ID

Formato de la Tabla STOCK

Nombre de Campo	Definición del campo	Comentarios
S_I_ID	200,000 Ids únicos	Se pueblan 100,000 por almacén
S_W_ID	2*W Ids únicos	
S_QUANTITY	numérico, 4 dígitos	
S_DIST_01	texto fijo, tamaño 24	
S_DIST_02	texto fijo, tamaño 24	
S_DIST_03	texto fijo, tamaño 24	
S_DIST_04	texto fijo, tamaño 24	
S_DIST_05	texto fijo, tamaño 24	
S_DIST_06	texto fijo, tamaño 24	
S_DIST_07	texto fijo, tamaño 24	
S_DIST_08	texto fijo, tamaño 24	
S_DIST_09	texto fijo, tamaño 24	
S_DIST_10	texto fijo, tamaño 24	
S_YTD	numérico, 8 dígitos	
S_ORDER_CNT	numérico, 4 dígitos	
S_REMOTE_CNT	Numérico, 4 dígitos	
S_DATA	texto variable, tamaño50	Da información

Clave primaria: (S_W_ID, S_I_ID)

(S_W_ID) Clave foránea, referencia a (W_ID)

(S_I_ID) Clave foránea, referencia a (I_ID)

1.4 Reglas de implementación

1.4.1 Se permite la agrupación física de registros dentro de la base de datos.

1.4.2 No se permite el uso de una vista que represente las filas para ahorrarse lecturas/escrituras lógicas.

Comentario: El motivo de esta cláusula es asegurar que la aplicación implementa el número de operaciones lógicas definidas en los perfiles de transacción sin combinar múltiples operaciones en una, mediante el uso de una vista.

1.4.3 Todas las tablas deben tener el número de filas adecuadamente escalado como se define en los requisitos de población de la base de datos (ver Cláusula 4.3).

1.4.4 Se permite la partición horizontal de las tablas pudiéndose asignar grupos de columnas de una tabla a diferentes archivos, discos o áreas. Si se implementa este método, los detalles de dicha partición deben ser puestos en conocimiento.

1.4.5 Se permite la partición vertical de las tablas pudiéndose asignar grupos de atributos (columnas) de una tabla a ficheros, discos o áreas diferentes a las que almacenen al resto de atributos. Si se implementa este método, los detalles de dicha partición deben ser puestos en conocimiento (ver cláusula 1.4.9 para limitaciones)

Comentario: Se permite la réplica de todas las tablas debiendo cumplir todas las copias los requisitos de atomicidad, consistencia y aislamiento que se definen en la cláusula 3. Si se utiliza este método, los detalles de dicha replica deben ser puestos en conocimiento.

1.4.6 Se permite la réplica de todas las tablas. Todas las copias de las tablas replicadas deben cumplir todos los requisitos de atomicidad, consistencia y aislamiento, tal y como se define en la cláusula 3. Si se implementa, los detalles de la réplica deben declararse.

Comentario: Solo necesita cumplir los requisitos de durabilidad definidos en la cláusula una de las copia de una tabla replicada.

1.4.7 Se pueden añadir o duplicar atributos de una tabla a otra siempre y cuando esos cambios no mejoren el rendimiento.

1.4.8 Cada atributo, según se describe en la cláusula 1.3.1 debe ser discreto en su estructura lógica y accesible de forma independiente por el gestor de datos. Por ejemplo W_STREET_1 y W_STREET_2 no pueden ser implementados como dos partes del atributo discreto W_STREET.

1.4.9 Cada atributo, según se describe en la cláusula 1.3.1 debe ser accesible como un atributo único por gestor de datos. Por ejemplo, S_DATA no puede implementarse como dos atributos discretos S_DATA_1 y S_DATA_2. Los atributos siguientes son excepciones de esta cláusula. No puede definirse una partición vertical entre los dos atributos usados para implementar estas excepciones:

- Se pueden implementar todos los atributos que contienen un valor de fecha y hora (ej. C_SINCE, H_DATE, O_ENTRY_D, y OL_DELIVERY_D) como

una combinación de dos atributos: un atributo de fecha y un atributo de hora.

- Se puede implementar el atributo C_DATA como dos atributos distintos del mismo tamaño y con el mismo tipo de datos.

1.4.10 La llave primaria de cada tabla no debe representar directamente la dirección física de las filas o cualquier desplazamiento sobre la misma. La aplicación no puede hacer referencia a las filas usando direccionamiento relativo ya que son simples desplazamientos sobre el comienzo del espacio de almacenamiento. Esto no excluye el uso de esquemas enlazados u otra organización de archivos que tengan funciones de adicción, eliminación y modificación de registros durante el transcurso del proceso. Excepción: la tabla History puede usar direccionamiento relativo pero se aplican los requerimientos restantes.

Comentario 1: Esta cláusula tiene como objetivo evitar que el programa de aplicación que ejecuta la transacción, o presenta una petición de transacción (ver cláusula 2.1.7), use identificadores físicos, en lugar de identificadores lógicos para todos los accesos, o que contenga código escrito por el usuario que transforme o ayude a transformar una llave lógica en la posición de la tabla de la fila o filas asociadas a esa llave. Por ejemplo, no se permite que la aplicación construya una tabla de transformación de direcciones lógicas a físicas y que la use para aumentar el rendimiento.

Comentario 2: Se puede usar registros internos o identificadores de fila, como por ejemplo, identificadores de tupla o cursores bajo las siguientes condiciones:

1. Para cada transacción ejecutada, el acceso a cualquier fila debe ser mediante la llave especificada en el perfil de transacción y no otro atributo. Con acceso se entiende la inserción, la eliminación, la lectura y la actualización de cualquier fila.

2. No puede violarse La cláusula 1.4.10.

1.4.11 Aunque no se realizan inserciones o eliminaciones en todas las tablas, no debe configurarse el sistema para tomar una ventaja especial de este hecho durante el test. A pesar de que las inserciones están intrínsecamente limitadas por el espacio de almacenamiento disponible en el sistema configurado, no debe haber restricción en la inserción, en cualquiera de las tablas, de un numero mínimo de filas equivalente al 5 % de la cardinalidad de la tabla y debe contarse con un valor de llave de al menos el doble de los valores de la llave presentes en esa tabla.

Comentario: Se requiere la configuración y valoración adecuada del espacio necesario para la cardinalidad adicional del 5 % (como espacio estático por la cláusula 4.2.3). En sistemas donde el espacio se configura y se asigna posteriormente de forma dinámica, se debe considerar este espacio como asignado e incluido en el presupuesto como espacio estático.

1.4.12 La precisión decimal mínima para cualquier cálculo realizado por el programa de aplicación debe ser igual a la precisión decimal máxima de cada uno de los elementos individuales de ese cálculo.

1.4.13 Se aplica a la implementación cualquier otra regla especificada en este documento (p.e. las reglas de consistencia de la cláusula 3.3).

1.5 Reglas de Integridad

1.5.1 Después de validar (commit) una transacción, los valores de la llave primaria deben ser únicos dentro de cada tabla. Por ejemplo, en el caso de la tabla con una parición vertical, los valores de la llave primaria de las filas de todas las particiones deben ser únicos.

1.5.2 Después de validar (commit)una transacción, no deben existir filas mal formadas en la base de datos. Una fila mal formada se produce cuando no se puede determinar el valor de algún atributo. Por ejemplo, en el caso de una partición vertical, una fila debe existir en todas las particiones.

1.6 Requisitos de Transparencia en el Acceso a los Datos

La transparencia en los accesos de datos es la propiedad del sistema por la cual se evita que el programa de aplicación tenga conocimiento de la localización de los datos particionados o el mecanismo de acceso a los mismos. Una implementación que use particiones verticales y horizontales debe cumplir los requisitos de transparente en los accesos de datos aquí descritos.

No hay ninguna serie finita de pruebas que pueda asegurar si un sistema aporta un acceso a los datos completamente transparente. Los requisitos siguientes describen las características mínimas necesarias para demostrar que el sistema ofrece un acceso transparente a los datos.

Comentario: La intención de esta cláusula que el acceso a los datos se produzca de forma transparente a través de una capa de aplicación comercial como pueden ser el gestor de datos/archivos (DBMS), el sistema operativo, el hardware o cualquier combinación de estos.

1.6.1 Cada una de las nueve tablas descritas en la cláusula 1.3 debe ser identificable por nombres que no tengan relación con las particiones de las tablas. Todas las operaciones de manipulación de datos en el programa de aplicación (ver cláusula 2.1.7) deben usar esos nombres exclusivamente.

1.6.2 El sistema debe evitar cualquier operación de manipulación de datos realizada usando los nombres descritos en al Cláusula 1.6.1 que daría como resultado una violación de las reglas de integridad (ver cláusula 1.5). Por ejemplo: el sistema debe evitar que una aplicación que no sea TPC-C valide la inserción de una fila en una tabla particionada verticalmente a menos que se haya insertado el resto de las particiones de esa fila.

1.6.3 Se puede permitir la manipulación de cualquier conjunto de filas o columnas a cualquier aplicación, que no sea TPC-C y que use los nombres que cumplen con la cláusula 1.6.1, siempre y cuando:

1. Sea identificable por cualquier condición arbitraria soportada por el DBMS sub-

yacente.

2. Use los nombres descritos en la cláusula 1.6.1 y la misma semántica y sintaxis en la manipulación de los datos de todas las tablas.

Por ejemplo se debe usar, la misma semántica y sintaxis para actualizar cualquier conjunto arbitrario de filas en una tabla que para actualizar otro conjunto de filas en otra tabla.

Comentario: la intención es que el programa de aplicación del TPC-C use mecanismos de manipulación de datos de propósito general en la base de datos.

Cláusula 2: Perfiles de Transacción y de Terminal

2.1. Definición de términos

2.1.1 En adelante se entiende como *seleccionar* a la acción de identificar (p.e referenciar, apuntar) una fila (o filas) sin obtener los datos que contiene.

2.1.2 En adelante se entiende como *recuperar* a la acción de acceder a un atributo de la base de datos y pasar su contenido al programa de aplicación.

Nota: los campos que corresponden a atributos de la base de datos se representan en MAYÚSCULAS. Los campos que se usan en el SUT o en el RTE, para cálculos o para comunicaciones con el terminal están representados en letra minúscula cursiva.

2.1.3 En adelante el término *transacción de base de datos* representa una operación sobre la base de datos que cumple plenamente con las propiedades ACID descritas en la Cláusula 3. Una *transacción comercial* está compuesta de una o varias transacciones de base de datos. Cuando se use solo el término *transacción* se refiere a una transacción comercial.

2.1.4 El símbolo [x..y] representa un intervalo cerrado de valores entre x e y.

2.1.5 Se califica como 'seleccionado aleatoriamente entre [x..y] al elemento seleccionado al azar de forma independientemente y con distribucion uniforme entre x e y. con una media de (x+y)/2, y con el mismo número de dígitos de precisión que se muestran. Por ejemplo, [0.01..100.00] tiene 10,000 valores, mientras que [1..100] tiene 100 valores.

2.1.6 Se califica como 'aleatorio no uniforme' al numero seleccionado aleatoria e independientemente siguiendo una distribución no uniforme entre el rango de valores [x..y] especificado. Se usa exclusivamente para generar números cliente, nombres de cliente y números de artículo. Este número debe generarse usando la función NURand la cual proporciona posiciones dentro del rango [x..y]. El resultado de la función puede utilizarse para obtener un nombre o un numero valido.

NURand(A, x, y)= (((random(0,A)| random(x,y))-C) %(y-x+1)+x

Donde:

- exp-1 | exp-2 representa la operación lógica OR bit a bit entre exp1 y exp2
- exp-1 % exp-2 representa el resto resultante de exp1/exp2.
- Random(x,y) significa seleccionado aleatoriamente dentro de [x..y]
- 'A' es una constante escogida de acuerdo con el tamaño del intervalo [x..y]
  - para C_LAST, el rango es [0..999] y A=255
  - para C_ID el rango es [1..3000] y A=1023
  - para OL_I:ID el rango es [1..100000] y A=8191

- 'C' es una constante de tiempo de ejecución escogida al azar entre [0..A] que puede variar sin alterar el rendimiento. Se debe usar el mismo valor de C para cada campo (C_LAST, C_ID y OL_I_ID) en todos los terminales emulados.

2.1.6.1 Para que el valor de C para C_LAST no altere el rendimiento debe cumplirse lo siguiente:

- Sea 'C-Load' el valor de 'C' para generar C_LAST cuando se pueble la base de datos.
  'C-Load' es un valor en el rango de [0..255] incluyendo 0 y 255.
- Sea 'C-Run' el valor de 'C' usado para generar C_LAST para la ejecución de la medida.
- Sea C-Delta el valor absoluto de la diferencia entre C-Load y C-Run. C-Delta debe ser un valor dentro del rango [65..119] incluyendo los valores de 65 y 119 y excluyendo los valores de 96 y 112

2.1.7 Se entiende como *programa de aplicación* al código que no forma parte de los componentes del sistema disponibles en el mercado sino que se ha escrito específicamente para implementar los perfiles de transacción (Cláusulas 2.4.2, 2.5.2, 2.6.2, 2.7.4) de este benchmark. Por ejemplo, los procedimientos de almacenamiento, los *disparadores* y las restricciones para la integridad referencial se consideran como parte del programa de aplicación cuando se usan para implementar alguna parte del perfil de transacción, sin embargo no se consideran como parte del programa de aplicación cuando sólo se usan para hacer cumplir las reglas de integridad (ver Cláusula 1.5) o los requisitos de transparencia (ver Cláusula 1.6) independientemente del perfil de transacción.

2.1.8 Se denomina *terminal* al dispositivo de interfaz capaz de introducir y mostrar caracteres desde y hacia un usuario con una pantalla mínima de 24x80. Se define terminal como los componentes que facilitan al usuario final la introducción de datos y la muestra de resultados. El terminal no debe tener ningún conocimiento de la aplicación excepto del formato, del tipo y de la posición de los campos.

2.2 Requisitos generales para el Terminal de Entrada/Salida

2.2.1 Aspectos sobre pantalla de entrada

2.2.1.1 El patrocinador del test debe reproducir la distribución de las pantallas de estrada/salida lo mas fielmente posible a lo descrito en la cláusulas 2.4.3.1, 2.5.3.1, 2.6.3.1, 2.7.3.1 y 2.8.3.1, informando de cualquier variación.

2.2.1.2 Se pueden modificar las pantallas de entrada/salida pueden suplir las limitaciones de la implementación siempre y cuando con ello no se vea mejorado el rendimiento. Se aplican las siguientes reglas:

1. Es posible traducir los títulos a cualquier lenguaje.

2. No es posible alterar los contenidos semánticos.

3. No es posible alterar el número de campos individuales.

4. No es posible decrementar el número de caracteres dentro de un campo (ej: la anchura del campo).

5. Es posible la reordenación y la redisposición de los campos.

6. Debe incluirse una copia de las nuevas especificaciones de pantalla y su distribución en el Informe Completo de Especificaciones (Full Disclosure Report).

2.2.1.3 Los campos que contienen cantidades monetarias están diseñados para la moneda Americana pudiéndose modificar para adaptarse a las representaciones de otras monedas (ej: usar otro símbolo de moneda, mover el punto decimal al menos un dígito a la derecha).

2.2.1.4 No es necesario mostrar en las pantallas de entrada/salida los campos (o grupo de campos) que no se estén utilizando. Por ejemplo, cuando una orden tiene menos de 15 artículos, los campos en la pantalla de entrada/salida correspondientes a los restantes artículos no se utilizan y no es necesario mostrarlos una vez que se hayan limpiado.

2.2.1.5 Se deben limpiar al principio de la transacción aquellos campos que puedan cambiar, incluso si el terminal selecciona consecutivamente el mismo tipo de transacción. Los campos deben limpiarse imprimiendo en ellos espacios o ceros.

Comentario: En el caso de que el patrocinador del test no crea conveniente el uso de espacios y ceros como caracteres para limpiar los campos, puede utilizar otro tipo de carácter con la condición de siempre se utilice el mismo carácter para limpiar un campo dado.

2.2.1.6 Para seleccionar el siguiente tipo de transacción se utiliza un menú. El menú, se compone de una o más líneas que deben mostrarse en el extremo superior o extremo inferior de la pantalla de entrada/salida. Si se necesita un campo de entrada para introducir la selección del menú, debe estar situado en las líneas reservadas para el menú.

2.2.1.7 El menú debe mostrar texto explícito (es decir, debe contener el nombre completo de cada transacción, y el procedimiento a seguir para seleccionar cada transacción). Deben mostrarse un mínimo de 60 caracteres (excluyendo espacios).

2.2.1.8 Se debe describir y la funcionalidad y el propósito de cualquier campo adicional que no corresponda con los indicados en las especificaciones de las pantallas de entrada/salida para cada tipo de transacción.

2.2.2 Introducción y Muestra de los campos

2.2.2.1 Se dice que un se ha introducido un campo una vez que el terminal emulado ha comunicado al SUT todos los caracteres significativos que componen los datos de entrada de dicho campo.

2.2.2.2 Se dice que se ha mostrado un campo una vez que el SUT ha comunicado al terminal emulado todos los caracteres significativos que componen los datos de dicho campo.

2.2.2.3 No se requiere la transmisión de un numero especifico de bytes para la comunicación de la entrada y salida. Se permite el uso de métodos para optimizar esta comunicación, tales como la compresión de mensajes.

2.2.2.4 El usuario emulado deberá proporcionar las siguientes características:

1. Los caracteres de entrada aparecen en la pantalla de entrada/salida (es decir, se muestran) cuando se teclean. Si no hay retardos perceptibles se puede comprobar este requisito visualmente. En caso contrario, se requiere que la aparición de los caracteres se verifique mediante medidas reales. Por ejemplo, se puede utilizar un analizador de protocolo, un medidor en el RTE, etc, para demostrar que el tiempo de respuesta del eco es menor de un segundo. No es necesario dicha verificación en el caso de utilizar dispositivos de eco local o de modo bloque.

   Comentario: una implementación que utilize un navegador web, un terminal o un PC emulando un terminal ya sea en eco local o en modo bloque, cumplirá el requisito de tiempo de respuesta menor de un segundo, así que no se necesita un test de eco.

2. Sólo se permite la introducción de datos en las posiciones de los campos de entrada de la pantalla de entrada/salida, es decir, los campos de salida, las etiquetas, y los espacios en blanco deben protegerse contra la introducción de datos.

3. Los campos dedicados a la entrada deben diseñarse de forma que puedan identificarse claramente, por ejemplo, utilizando areas resaltadas, subrayado, video inverso, divisores de columna, etc).

4. Los campos sólo pueden contener caracteres significativos. En los campos alfanuméricos las posiciones vacías deben rellenarse con espacios en blanco o valores nulos. En los campos numéricos, las entradas de longitud menor al máximo de dígitos permitidos deben justificarse a la derecha en la pantalla de salida.

5. Los campos necesarios para que se complete la aplicación deben contener algún valor antes de comenzar a medir el TR de transacción o la aplicación debe poseer un conjunto de rutinas de manejo de error que informen al usuario que dichos campos no se han introducido.

6. Los campos pueden introducirse y editarse en cualquier orden. Específicamente:

   - El usuario emulado debe ser capaz de mover el cursor hacia adelante y hacia atrás.
   - La aplicación no puede depender de que los campos sean introducidos en un orden particular.
   - El usuario puede retornar a un campo que ha sido introducido y cambiar su valor antes de comenzar la medida del TR de transacción.

7. Los campos numéricos deben estar protegidos de entradas no numéricas. Si se introduce uno o más caracteres no numéricos en uno de estos campos se deberá informar al usuario de un error en la entrada de datos.

Comentario: la comprobación de los datos de entrada puede realizarse tanto por parte del terminal como por parte de la aplicación o por una combinación de ambos y debe realizarse antes de comenzar la transacción de base de datos. Por norma, un dato de entrada no válido no puede provocar la cancelación de una transacción.

2.2.2.5 Todos los campos cuyos valores son actualizados por la transacción comercial en curso deben mostrar el valor nuevo.

2.3 Requisitos Generales Para Los Perfiles De Transacción

Cada transacción debe implementarse de acuerdo con los perfiles de transacción especificados. Además:

2.3.1 El orden de la manipulación de datos dentro de los límites de transacción es irrelevante (a no ser que se especifique lo contrario, ver Cláusula 2.4.2.3) con tal de que la implementación de las transacciones sea equivalente funcionalmente a lo especificado en el perfil de transacción.

2.3.2 Los perfiles de transacción especifican los requisitos mínimos de recuperación y actualización de datos. Debe informarse de la introducción de operaciones adicionales de recorrido de la base de datos y de operaciones adicionales de manipulación dentro de la transacción de base de datos, así como dar una explicación de la función de dichas operaciones.

2.3.3 Cada atributo debe obtenerse de la tabla designada en el perfil de transacción.

Comentario: Esta cláusula tiene como intención evitar la reducción del número de operaciones lógicas en la base de datos necesarias para implementar cada transacción.

2.3.4 No se puede realizar ninguna operación de manipulación de datos del perfil de transacción antes de que se hayan comunicado todos los datos al terminal, o después de que el SUT haya comunicado algún dato al terminal emulado.

Comentario: La intención de esta cláusula es asegurar que para una transacción de negocios dada ninguna operación de manipulación del perfil de transacción se realiza antes del sello de hora tomado al principio del TR de transacción o después del sello de hora tomado al final del TR de transacción (ver Cláusula 5.3).

2.3.5 Si las transacciones se encaminan u organizan en el SUT, se hace necesario el uso de un monitor de procesado de transacciones o un software equivalente disponible en el mercado (en adelante MT, Monitor de Transacciones) con las siguientes características/capacidades:

Operación - El MT debe permitir:

- La gestión de prioridad de las peticiones/servicios.
- El multiplexado/demultiplexado de peticiones/servicios.
- Balanceado automático de carga de trabajo.
- Recepción, encolado y ejecución de múltiples solicitudes/servicios de forma concurrente.

Seguridad - El MT debe permitir:

- La capacidad de comprobar y autorizar la ejecución de cada servicio en el momento de la solicitud del servicio.
- La restricción de funciones administrativas a los usuarios autorizados.

Administración/Mantenimiento - EL MT debe tener la capacidad de realizar el manejo de los recursos del MT de una manera centralizada, no programada y con una configuración dinámica, incluyendo hardware, software, servicios, reglas para la gestión de prioridad de la cola, etc.

Recuperación - El TM debe tener la capacidad de:

- Enviar códigos de error a una aplicación.
- Detectar y terminar transacciones de larga ejecución basándose en intervalos de tiempo máximo predefinidos.

Transparencia de Aplicación - El contexto/s de mensajes que existen entre el cliente y los programas de servidor de aplicación deben ser manejados solamente por el MT. El cliente y los programas de servidor de aplicación no deben tener ningún conocimiento de los contextos de mensajes o de mecanismos de comunicación que soportan ese contexto.

Comentario 1: Los siguientes son ejemplos de implementaciones que no cumplen con los requisitos de Transparencia de Aplicación.

1. El cliente y los programas de servidor de aplicación usan el mismo identificador para mantener el contexto de mensaje para múltiples transacciones.
2. Se requiere el cambio y/o recompilación de los programas de aplicación de cliente y/o servidor cuando por la administración del MT se cambie el número de colas o estructuras de datos equivalentes utilizadas por el MT para mantener el contexto de los mensajes entre los programas de aplicación de cliente y de servidor.

Comentario 2: La intención de esta cláusula es exigir el uso los monitores de transacciones de propósito general disponibles en el mercado, y excluir el software de propósito especial desarrollado para pruebas de benchmark u otros usos limitados. Es un hecho que la implementación de las características y funcionalidades descritas anteriormente varían de un vendedor a otro. Tales diferencias no excluyen el cumplimiento de los requisitos de esta cláusula.

Comentario 3: El MT o el software equivalente no es necesario si el DBMS mantiene un contexto individual para cada terminal emulado.

2.3.6 Cualquier error que causara una transacción TPC-C no válida debe ser detectado e informado. Se incluyen en las transacciones TPC-C no válidas aquellas que si se confirmaran (se realiza un COMMIT) violarían el nivel de consistencia de la base de datos definido en la Cláusula 3. Se debe cancelar dichas transacciones (realizar un ROLLBACK). La detección de una transacción no válida debe informarse al usuario en la pantalla de salida o en el caso de la parte aplazada de la transacción de reparto, en la bitácora de reparto.

Comentario 1: Algunos ejemplos de errores que podrían provocar una transacción no válida son:

- Selección o actualización de un registro no existente.
- Fallo en la inserción de un nuevo registro.
- Fallo en la eliminación de un registro existente.
- Fallo en la selección o actualización de un registro existente.

Comentario 2: No se ha definido la información exacta a ofrecer en el momento de producirse un error sino que es especifico de cada implementación.

2.4 Transacción New-Order

La transacción comercial New Order consiste en introducir una orden completa por medio de una única transacción de base de datos. Representa una transacción de carga media, de lectura/escritura con alta frecuencia de ejecución y estrictos requisitos en cuanto al tiempo de respuesta para satisfacer a los usuarios en línea. Esta transacción es la espina dorsal del benchmarck. Está diseñada para ofrecer una carga variable al sistema y así reflejar la actividad en línea de la base de datos como la que puede encontrar en un ambiente de producción tipico.

2.4.1 Generación de los datos de entrada

2.4.1.1 Para un terminal dado el número de almacén (W_ID) es constante durante todo el intervalo de medida (ver Cláusula 5).

2.4.1.2 El número de distrito (D_ID) se selecciona aleatoriamente entre [1..10] dentro del almacén local (D_W_ID=W_ID). El número de cliente (C_ID) se selecciona aleatoriamente con distribución no uniforme utilizando la función NURand(1023,1,3000) para el número de distrito seleccionado (C_D_ID=D_ID) y para el número de almacén local (C_W_ID=W_ID).

2.4.1.3 El número de artículos de la orden (ol_cnt) se selecciona aleatoriamente entre [5..15] (con una media de 10). Este campo no se introduce sino que se genera en el emulador de terminal para determinar el tamaño de la orden. O_OL_CNT se muestra posteriormente tras ser calculado por el SUT.

2.4.1.4 Un 1 % de las transacciones New-Order se eligen de forma aleatoria para simular entradas de datos erróneos por parte del usuario y probar la cancelación transacciones. Esto puede implementarse generando un número aleatorio entre [1..100].

Comentario: Todas las transacciones New-Order deben tener datos de entrada generados de forma independiente. Los datos de entrada de una transacción se ha cancelado no se pueden utilizar para una siguiente transacción.

2.4.1.5 Para cada uno de los ol_cnt artículos en la orden:

1. Se selecciona un número de artículo (OL_I_ID) aleatoriamente con distribución no uniforme utilizando la función NURand(8191,1,100000). Si es el último artículo de la orden y rbk=1 se utiliza un número de artículo no válido.

   Comentario: Un valor no válido para un número de artículo es un valor que no se encuentre en la base de datos tal que su uso produzca una condición de "no encontrado"en el programa de aplicación. Esta condición debe provocar la cancelación (ROLLBACK) de la transacción de base de datos en curso.

2. El número de almacén de suministro (OL_SUPPLY_W_ID) corresponde al almacén local el 99 % de los casos y a un almacén remoto en el 1 % de los casos. Esto puede implementarse generando un número aleatorio entre [1..100].

   - Si x>1 el artículo proviene del almacén local (OL_SUPPLY_W_ID=W_ID)

- Si x=1 el artículo proviene de un almacén remoto.
  (OL_SUPPLY_W_ID) se selecciona dentro del rango de almacenes activos excluyendo W_ID.

Comentario 1: Suponiendo una media de 10 artículos por orden, aproximadamente el 90 % de todas las órdenes pueden cubrirse completamente por los stocks del almacén local.

Comentario 2: Si el sistema está configurado para un único almacén, todos los artículos provienen del almacén local.

3. La cantidad (OL_QUANTITY) se selecciona aleatoriamente entre [1..10].

2.4.1.6 La fecha de entrada de orden (O_ENTRY_DATE) se genera en el SUT utilizando la fecha actual del sistema.

2.4.1.7 Una línea de orden (o artículo) se dice que es local, si es el almacén local el que suministra ese articulo, es decir si OL_SUPPLY_W_ID = O_W_ID.

2.4.1.8 Una línea de orden se dice que es remota, cuando un almacén remoto es el que suministra ese articulo, es decir cuando OL_SUPPLY_W_ID no es igual a O_W_ID.

2.4.2 Perfil de Transacción

2.4.2.1 La introducción de una New-Order se realiza mediante una única transacción de base de datos. A continuación se detallan las operaciones realizadas en esta transacción.

2.4.2.2 Dado un número de almacén (W_ID), un número de distrito (D_W_ID, D_ID), un número de cliente (C_W_ID, C_D_ID, C_ID), una cantidad de artículos (ol_cnt, no comunicado al SUT), y una serie de artículos (OL_I_ID), almacenes de suministro
(OL_SUPPLY_W_ID), y cantidades (OL_QUANTITY):

- Los datos de entrada (ver Cláusula 2.4.3.2) se comunican al SUT.
- Se comienza una transacción de base de datos.
- Se selecciona la fila en la tabla WAREHOUSE que coincide con W_ID, y se recupera la tasa de impuesto de almacén W_TAX.
- Se selecciona la fila de la tabla WAREHOUSE que coincide con D_W_ID y con D_ID, se recupera la tasa de impuesto del distrito D_TAX, y el siguiente número de orden disponible para ese distrito D_NEXT_O_ID se recupera y se incrementa en una unidad.
- Se selecciona la fila en la tabla CUSTOMER que coincide con C_W_ID, C_D_ID y C_ID, y se recuperan la tasa de descuento del cliente C_DISCOUNT, el último nombre del cliente C_LAST y el estado de crédito del cliente C_CREDIT.

- Se inserta una nueva fila tanto en la tabla NEW_ORDER como en la tabla ORDER para registrar la creación de una New-Order. Se introduce en O_CARRIER_ID un valor nulo. Si la orden incluye solamente líneas de artículo locales, se introduce un 1 en O_ALL_LOCAL, en caso contrario se introduce un 0.
- Se calcula O_OL_CNT para que alcance ol_cnt.
- Para cada uno de los O_OL_CNT artículos en la orden:
  - Se selecciona en la tabla ITEM la fila que coincide con I_ID (igual a OL_I_ID) y se recuperan el precio del artículo I_PRICE, el nombre del artículo I_NAME e I_DATA. Si I_ID contiene un valor no válido (ver Cláusula 2.4.1.5), se dará la condición de "no encontrado", provocando la cancelación (ROLLBACK) de la transacción.
  - Se selecciona en la tabla STOCK la fila que coincide con S_ID (igual a OL_I_ID) y con S_W_ID (igual a OL_SUPLLY_W_ID). Se recupera la cantidad en stock S_QUANTITY, S_DIST_XX, donde XX corresponde al número de distrito y S_DATA. Si el valor recuperado de S_QUANTITY excede a OL_QUANTITY en 10 unidades o más, entonces se decrementa S_QUANTITY en OL_QUANTITY unidades; en caso contrario se actualiza S_QUANTITY de la forma (S_QUANTITY - OL_QUANTITY)+91. Se incrementa S_YTD en OL_QUANTITY unidades y S_ORDER_CNT se incrementa en uno. Si la línea de orden es remota entonces S_REMOTE_CNT se incrementa en uno.
  - La suma para cada artículo de la orden (OL_AMOUNT) se calcula como: OL_AMOUNT=OLQUANTITY*I_PRICE
  - Se examina las cadenas I_DATA y S_DATA. Si ambas contienen la cadena 'ORIGINAL' el campo *brand-generic* de ese artículo se inserta una 'B', en caso contrario se inserta una 'G'.
  - Se inserta una nueva fila en la tabla ORDER-LINE para registrar el artículo en la orden. Se inserta un valor nulo en OL_DELIVERY. En OL_NUMBER se introduce un valor único de entre todas las filas de la tabla ORDER-LINE que tienen el mismo valor de OL_O_ID. Se introduce en OL_DIST_INFO el contenido de S_DIST_xx, donde xx corresponde al número de distrito (OL_D_ID).
- La suma total *total_amount* de la orden completa se calcula como: Sum(OL_AMOUNT)*(1-C_DISCOUNT)*(1+W_TAX+D_TAX)
- Se confirma la transacción (COMMIT), de no haberse cancelado (ROLLBACK) a causa de un valor no valido en el ultimo número de articulo.
- Se comunican los datos de salida al terminal.

2.4.2.3 En las transacciones que se han cancelado a causa de un número de artículo inválido, el perfil de transacción debe ejecutarse completamente con la excepción de que los siguientes pasos no necesitan realizarse:

- Seleccionar y recuperar la fila de la tabla de STOCK utilizando un S_I_ID igual al articulo no valido.
- Examinar la cadena I_DATA y S_DATA utilizando el numero de artículo no valido.
- Insertar una nueva fila en la tabla ORDER_LINE utilizando el numero de artículo valido.
- Añadir a OL_AMOUNT la suma para el artículo no usado.

La transacción no se confirma (COMMIT) sino que se cancela (ROLLBACK).

Comentario 1: La intencion de esta cláusula es asegurar que se procesen todos los artículos validos de la transacción New-Order antes de procesar el articulo no valido. El conocimiento de que hay un artículo no válido, solo puede utilizarse para evitar la ejecución de los pasos anteriores. No se puede utilizar esta información para cualquier otra optimización, como por ejemplo evitar realizar otros pasos, cambiar la ejecución de otros pasos, usar un tipo diferente de transacción, etc.

Comentario 2: Esta cláusula es una excepción de la Cláusula 2.3.1. El orden de la manipulación de los datos antes de producirse la condición de *no encontrado* es irrelevante.

2.4.3 Terminal de Entrada Salida

2.4.3.1 En cada transacción el terminal de origen debe mostrar la siguiente pantalla de entrada/salida con todos los campos de entrada/salida limpios (bien utilizando espacios o ceros), excepto el campo de almacén que no cambia y que debe mostrar el valor fijo de W_ID asociado con el terminal.

```
         1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890

                                  New Order
Warehouse: 9999   District: 99                    Date: DD-MM-YYYY hh:mm:ss
Customer:  9999   Name: XXXXXXXXXXXXXXXX   Credit: XX   %Disc: 99.99
Order Number: 99999999  Number of Lines: 99        W_tax: 99.99   D_tax: 99.99

 Supp_W  Item_Id  Item Name                 Qty  Stock  B/G  Price    Amount
  9999   999999   XXXXXXXXXXXXXXXXXXXXXXXX  99    999    X   $999.99  $9999.99
  9999   999999   XXXXXXXXXXXXXXXXXXXXXXXX  99    999    X   $999.99  $9999.99
  9999   999999   XXXXXXXXXXXXXXXXXXXXXXXX  99    999    X   $999.99  $9999.99
  9999   999999   XXXXXXXXXXXXXXXXXXXXXXXX  99    999    X   $999.99  $9999.99
  9999   999999   XXXXXXXXXXXXXXXXXXXXXXXX  99    999    X   $999.99  $9999.99
  9999   999999   XXXXXXXXXXXXXXXXXXXXXXXX  99    999    X   $999.99  $9999.99
  9999   999999   XXXXXXXXXXXXXXXXXXXXXXXX  99    999    X   $999.99  $9999.99
  9999   999999   XXXXXXXXXXXXXXXXXXXXXXXX  99    999    X   $999.99  $9999.99
  9999   999999   XXXXXXXXXXXXXXXXXXXXXXXX  99    999    X   $999.99  $9999.99
  9999   999999   XXXXXXXXXXXXXXXXXXXXXXXX  99    999    X   $999.99  $9999.99
  9999   999999   XXXXXXXXXXXXXXXXXXXXXXXX  99    999    X   $999.99  $9999.99
  9999   999999   XXXXXXXXXXXXXXXXXXXXXXXX  99    999    X   $999.99  $9999.99
  9999   999999   XXXXXXXXXXXXXXXXXXXXXXXX  99    999    X   $999.99  $9999.99
  9999   999999   XXXXXXXXXXXXXXXXXXXXXXXX  99    999    X   $999.99  $9999.99
  9999   999999   XXXXXXXXXXXXXXXXXXXXXXXX  99    999    X   $999.99  $9999.99
Execution Status: XXXXXXXXXXXXXXXXXXXXXXXX                   TOtal:  $99999.99
```

2.4.3.2 El usuario emulado debe introducir los datos de entrada necesarios en los campos apropiados de la pantalla E/S, que se dividen en dos grupos organizados de la forma:

- Dos campos D_ID y C_ID.

  Comentario: No se puede introducir el valor de *ol_cnt* si debe determinarse en la aplicación tras procesar los datos de entrada.

- Un grupo de campos repetidos: OL_I_ID, OLSUPLLY_W_ID y OL_QUAMTITY. El grupo se repite .ºl_cnt"veces (una por cada artículo de la orden). Los valores de esos campos se determinan según se especifica en la cláusula 2.4.1.5.

  Comentario: Para mantener una cantidad razonable de entrada por teclado se deben rellenar los campos de almacén de suministro para cada articulo, incluso si el almacén de suministro es el almacén local.

2.4.3.3 El terminal emulado debe mostrar, en los campos apropiados de la pantalla E/S, todos los datos de entrada y de salida resultantes de la ejecución de la transacción. Los campos mostrados se dividen en dos grupos de la forma:

- Un grupo de campos no repetidos: W_ID, D_ID, O_ID, O_OL_CNT, C_LAST, C_CREDIT, C_DISCOUNT, W_TAX, D_TAX, O_ENTRY:D, *total_amount*, y un mensaje opcional del estado de la ejecución además de *El número de artículo no es válido*.

- Un grupo de campos repetidos: OL_SUPPLY_W_ID, OL_I_ID, I_NAME, OL_QUANTITY, S_QUANTITY, *brand-generic*, I_PRICE, y OL_AMOUNT. El grupo se repite O_OL_CNT veces (una por cada artículo de la orden), igual al valor de ol_cnt calculado.

2.4.3.4 En las transacciones que se han cancelado a causa de un número de artículo no válido, el terminal emulado debe mostrar en los campos apropiados de la pantalla E/S: W_ID, D_ID, C_ID, C_LAST, C_CREDIT, O:ID, y el mensaje del estado de la ejecución *El número de artículo no es válido*. No es necesario imprimir un mensaje del estado de la ejecución si la transacción se realiza satisfactoriamente.

Comentario: Se debe mostrar en O_ID el número de la orden rechazada, verificar la parte de la transacción que fue procesada.

2.5 Transacción Payment

La transacción comercial Payment actualiza el saldo del cliente y registra un pago en las estadísticas de venta del almacén y del distrito. Representa una transacción de carga baja, de lectura/escritura con una frecuencia de ejecución alta y unos requisitos estrictos en cuanto al tiempo de respuesta para satisfacer a los usuarios en línea. Además esta transacción incluye el acceso a la tabla CUSTOMER sin utilizar la llave primaria.

2.5.1 Generación de los datos de entrada

2.5.1.1 En cada terminal, el número de almacén local (W_ID) es constante durante todo el intervalo de medida.

2.5.1.2 El número de distrito (D_ID) se selecciona aleatoriamente entre [1..10] para el almacén local (D_W_ID=W_ID). El cliente seleccionado se selecciona el 60 % de las veces a través del su último nombre (C_W_ID,C_D_ID,C_LAST) y el 40 % de las veces a través de su número de cliente (C_W_ID,C_D_ID,C_ID). Independientemente del modo de selección, el almacén contra el que se ejecuta el pago es el almacén local el 85 % de las veces y el 15 % de las veces de selecciona un almacén remoto al azar. Se puede implementar lo anterior obteniendo dos números aleatorios x e y entre [1..100].

- Si x<=85 se selecciona un cliente con números de almacén y distrito iguales a los seleccionados (C_D_ID=D_ID).
- Si x>85 se selecciona un cliente de un distrito escogido aleatoriamente (C_D_ID se selecciona al azar entre [1..10]) y de almacén escogido aleatoriamente (C_W_ID se selecciona al azar entre del rango de almacenes activos (ver Cláusula 4.2.2). El cliente está realizando un pago a través de un almacén y distrito diferentes a los suyos.
- Si y<=60 se genera el ultimo nombre de cliente (C_LAST) de acuerdo con la cláusula 4.3.2.3 utilizando un valor aleatorio con distribución no uniforme obtenido mediante la función NURand(255,0,999). El cliente está utilizando su último nombre para el pago, pudiendo haber varios clientes con el mismo ultimo nombre.
- Si y>60 se selecciona un número de cliente aleatorio (C_ID) mediante la función NURand(1023,1,3000). El cliente está utilizando su número de cliente para el pago.

Comentario: si el sistema está configurado para un único almacén, únicamente se seleccionan los clientes del almacén local.

2.5.1.3 La cantidad del pago (H_AMOUNT) se selecciona aleatoriamente entre [1,00..5000,00].

2.5.1.4 La fecha de pago (H_DATE) se genera en el SUT utilizando la fecha actual del sistema.

2.5.1.5 Se dice que la transacción Payment es local si el cliente pertenece al almacén contra el que se produce el pago (cuando C_W_ID=W_ID).

2.5.1.6 Se dice que la transacción Payment es remota si el almacén contra el que se ejecuta el pago no es al que pertenece el cliente (cuando C_W_ID no es igual a W_ID).

2.5.2 Perfil de transacción

2.5.2.1 La transacción Payment introduce el pago un cliente mediante una única transacción de base de datos. A continuación se detallan los pasos realizados en esta transacción.

2.5.2.2 Dado un número de almacén (W_ID), un número de distrito (D_W_ID,D_ID), un número de cliente (C_W_ID, C_D_ID,C_ID) o un último nombre de cliente (C_W_ID,C_D_ID,C_LAST), y una cantidad de pago (H_AMOUNT):

- Se comunican los datos al SUT.
- Se comienza una transacción de base de datos.
- Se selecciona la fila de la tabla de WAREHOUSE que coincide con W_ID. Se recuperan los campos W_NAME, W_STREET_1, W_STREET_2, W_CITY, W_STATE, y W_ZIP. El balance anual hasta la fecha del distrito D_YTD, se incrementa H_AMOUNT unidades.
- Caso 1, el cliente se selecciona utilizando el número de cliente:

  Se selecciona la fila de la tabla CUSTOMER que coincide con C_W_ID, C_D_ID y C_ID. Se recuperan los campos C_FIRST, C_MIDDLE, C_LAST, C_STREET_1, C_STREET_2, C_ZPI, C_CITY, C_STATE, C_PHONE, C_SINCE, C_CREDIT, C_CREDIT_LIM, C_DISCOUNT, y C_BALANCE. Se decrementa C_BALANCE en H_AMOUNT unidades. Se incrementa C_PAYMENT en una unidad.
- Caso 2, el cliente se selecciona utilizando su último nombre:

  Se seleccionan toas las filas de la tabla CUSTOMER que coinciden con C_W_ID, C_D_ID y C_LAST distribuidas en el orden ascendente de C_FIRST. Si n es el número de filas seleccionadas, se recuperan los campos C_ID, C_FIRST, C_MIDDLE, C_STREET_1, C_STREET_2, C_ZPI, C_CITY, C_STATE, C_PHONE, C_SINCE,C_CREDIT, C_CREDIT_LIM, C_DISCOUNT, y C_BALANCE de la fila que ocupa la posición n/2 (redondeando en exceso). Se decrementa C_BALANCE en H_AMOUNT unidades. Se incrementa C_PAYMENT una unidad.
- Si el valor de C_CREDIT es igual a 'BC' se recupera también el campo C_DATA del cliente seleccionado y se inserta la información historica contenida en los campos C_ID, C_D_ID, D_W_ID, D_ID, W_ID, y H_AMOUNT a la derecha de C_DATA desplazando a la izquierda el contenido de C_DATA un número de bytes igual a la logitud total de los campos introducidos, descartando los bytes que salen por el lado izquierdo del campo C_DATA. El contenido del campo C_DATA nunca debe exceder los 500 caracteres. Se actualiza el cliente con el nuevo campo C_DATA. Si el campo C_DATA se implementa utilizando dos campos (ver Cláusula 1.4.9), dichos campos deben tratarse como uno sólo.

  Comentario: EL formato utilizado para almacenar la información histórica debe ser tal que lo mostrado en la pantalla de E/S esté en un formato legible (ej: la

parte de W_ID de C_DATA debe usar el mismo formato que el campo de salida W_ID).

- H_DATA se construye concatenando W_NAME y D_NAME separados por 4 espacios.
- Se inserta una nueva fila en la tabla HISTORY con H_C_ID = C_ID, H_C_D_IC = C_D_ID, H_C_W_ID = C_W_ID, H_D_ID, y H_W_ID = W_ID.
- La transacción de base de datos se confirma (COMMIT).
- Los datos de salida se comunican al terminal.

2.5.3 El Terminal de Entrada/Salida

2.5.3.1 En cada transacción el terminal de origen debe mostrar la siguiente pantalla de entrada/salida con todos los campos limpios (bien utilizando espacios o ceros) con la excepción del campo de almacén que no cambia y debe mostrar el valor fijo de W_ID asociado con ese terminal. Además todos los campos de direcciones (ej: W_STREET_1, W_STREET_2, W_CITY, W_STATE y W_ZIP) del almacén pueden aparecer fijos en sus campos correspondientes si se han obtenido en una transacción anterior.

```
          1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
1                                  Payment
2  Date: DD-MM-YYYY hh:mm:ss
3
4  Warehouse: 9999                         District: 99
5  XXXXXXXXXXXXXXXXXXX                     XXXXXXXXXXXXXXXXXXX
6  XXXXXXXXXXXXXXXXXXX                     XXXXXXXXXXXXXXXXXXX
7  XXXXXXXXXXXXXXXXXXX XX XXXXX-XXXX       XXXXXXXXXXXXXXXXXXX XX XXXXX-XXXX
8
9  Customer: 9999  Cust-Warehouse: 9999  Cust-District: 99
10 Name:   XXXXXXXXXXXXXXX XX XXXXXXXXXXXXXXX     Since:  DD-MM-YYYY
11         XXXXXXXXXXXXXXXXXXX                    Credit: XX
12         XXXXXXXXXXXXXXXXXXX                    %Disc:  99.99
13         XXXXXXXXXXXXXXXXXXX XX XXXXX-XXXX      Phone:  XXXXXX-XXX-XXX-XXXX
14
15 Amount Paid:          $9999.99      New Cust-Balance: $-9999999999.99
16 Credit Limit:   $9999999999.99
17
18 Cust-Data: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
19            XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
20            XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
21            XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
22
23
24
```

2.5.3.2 El usuario emulado debe introducir en los campos apropiados de la pantalla E/S los datos de entrada necesarios. Los datos de entrada se obtienen a partir de los campos: D_ID, C_ID o C_LAST, C_D_ID, C_W_ID, y H_AMOUNT.

Comentario: Para mantener una cantidad razonable de entrada por teclado, se debe introducir el campo del almacén del cliente incluso cuando se trate del almacén local.

2.5.3.3 El terminal emulado debe mostrar en los campos apropiados de la pantalla de E/S, todos los datos de entrada y de salida resultantes de la ejecución de la transacción. Se muestran los siguientes campos: W_ID, D_ID, C_ID, C_D_ID, C_W_ID, W_STREET_1, W_STREET_2, W_CITY, W_STATE , W_ZIP, D_STREET_1, D_STREET_2, D_CITY, D_STATE , D_ZIP, C_FIRST, C_MIDDLE, C_LAST,

C_STREET_1, C_STREET_2, C_ZIP, C_CITY, C_STATE, C_PHONE, C _SINCE, C_CREDIT, C_CREDIT_LIM, C_DISCOUNT, C_BALANCE, los 200 primeros caracteres de C_DATA (sólo si C_CREDIT='BC'), H_AMOUNT, y H_DATE.

2.6 Transacción Order-Status

La transacción comercial Order Status realiza una consulta sobre el estado de la última orden de un cliente. Representa una transacción de peso medio, de solo lectura, con una frecuencia de ejecución baja y requisitos poco estrictos en cuanto al tiempo de respuesta para satisfacer a los usuarios en línea. Además Además esta transacción incluye el acceso a la tabla CUSTOMER sin utilizar la llave primaria.

2.6.1 Generación de datos de entrada

2.6.1.1 Dado un número de almacén (W_ID) constante durante todo el intervalo de medida.

2.6.1.2 Se selecciona aleatoriamente entre [1..10] el número de distrito (D_ID) para el almacén local. Se selecciona aleatoriamente un cliente en el 60 % de las veces utilizando su último nombre (C_W_ID, C_D_ID, C_LAST) y en el 40 % de las veces utilizando el número de cliente (C_W_ID, C_D_ID, C_ID) para el distrito (C_D_ID=D_ID) y el almacén local (C_W_ID=W_ID) seleccionados. Esto puede implementarse generando un número aleatorio 'y' entre [1..100]:

- Si y<=60 se genera un último nombre ('last name') de cliente de acuerdo con la cláusula 4.3.2.3 mediante un valor aleatorio utilizando la función NURand(255,0,999). El cliente utiliza su ultimo nombre, pudiendo coincidir con los de otros clientes en la base de datos.
- Si y>60 se selecciona aleatoriamente con distribución uniforme un número de cliente (C_ID) utilizando la función NURand(1023,1,3000). El cliente utiliza su número de cliente.

2.6.2 Perfil de transacción

2.6.2.1 La consulta del estado de una orden se lleva a cabo mediante una única transacción de base de datos. A continuación se detallan las pasos realizados en esta transacción.

2.6.2.2 Dado un número de cliente (C_W_ID, C_D_ID, C_ID):

- Los datos de entrada se comunican al SUT (ver Cláusula 2.6.3.2)
- Se comienza una transacción de base de datos.
- Caso 1, el cliente se selecciona por su número de cliente: se selecciona la fila de la tabla CUSTOMER que coincide con C_W_ID, C_D_ID y C_ID, y se recuperan los campos C_BALENCE, C_FIRST, C_MIDDLE y C_LAST.
- Caso 2, el cliente se selecciona por el último nombre de cliente: se seleccionan todas las filas de la tabla CUSTOMER que coincidan con C_W_ID, C_D_ID y C_LAST son y se ordenan en orden creciente de C_FIRST. Si 'n' es el número de filas seleccionadas, los campos C_BALENCE, C_FIRST, C_MIDDLE

y C_LAST se recuperan de la fila que ocupa la posición la posición n/2 (redondeando en exceso).

- Se seleccionan las filas de la tabla ORDER que coinciden con O_W_ID (igual a C_W_ID), O_D_ID (igual a C_D_ID), y con OL_O_ID (igual a O_ID), y se recuperan las series correspondientes de OL_I_ID, OL_SUPPLY_W_ID, OL_QUANTITY, OL_AMOUNT OL_DELIVERY.

- Se confirma (COMMIT) la transacción de base de datos.

- Los datos de salida se comunican al terminal.

2.6.3 El terminal de entrada salida

2.6.3.1 En cada transacción el terminal de origen debe mostrar la siguiente pantalla de entrada/salida con todos los campos limpios (bien utilizando espacios o ceros) exceptuando el campo de almacén que no cambia y debe mostrar el valor fijo de W_ID asociado con ese terminal.

```
         1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
1                                Order-Status
2  Warehouse: 9999   District: 99
3  Customer: 9999   Name: XXXXXXXXXXXXXXXX XX XXXXXXXXXXXXXXXX
4  Cust-Balance: $-99999.99
5
6  Order Number: 99999999   Entry-Date: DD-MM-YYYY hh:mm:ss   Carrier-Number: 99
7  Supp_W       Item_Id    Qty     Amount      Delivery-Date
8    9999       999999     99     $99999.99      DD-MM-YYYY
9    9999       999999     99     $99999.99      DD-MM-YYYY
10   9999       999999     99     $99999.99      DD-MM-YYYY
11   9999       999999     99     $99999.99      DD-MM-YYYY
12   9999       999999     99     $99999.99      DD-MM-YYYY
13   9999       999999     99     $99999.99      DD-MM-YYYY
14   9999       999999     99     $99999.99      DD-MM-YYYY
15   9999       999999     99     $99999.99      DD-MM-YYYY
16   9999       999999     99     $99999.99      DD-MM-YYYY
17   9999       999999     99     $99999.99      DD-MM-YYYY
18   9999       999999     99     $99999.99      DD-MM-YYYY
19   9999       999999     99     $99999.99      DD-MM-YYYY
20   9999       999999     99     $99999.99      DD-MM-YYYY
21   9999       999999     99     $99999.99      DD-MM-YYYY
22   9999       999999     99     $99999.99      DD-MM-YYYY
23
24
```

2.6.3.2 El usuario emulado debe introducir, en los campos apropiados de la pantalla de entrada/salida los datos de estrada necesarios: D_ID y bien C_ID o C_LAST.

2.6.3.3 El terminal emulado debe mostrar en los campos apropiados de la pantalla de E/S, todos los datos de entrada y los datos de salida resultantes de la ejecución de la transacción, dichos campos se dividen en dos grupos:

- Un grupo no repetido de campos: W_ID, D_ID, C_ID, C_FIRST, C_MIDDLE, C_LAST, C_BALANCE, O_ID, O_ENTRY y O_CARRIER_ID.

- Un grupo repetido de campos: OL_SUPPLY_W_ID, OL_I_ID, OL_QUANTITY, OL_AMOUN y OL_DELIVERY_D. El grupo se repite O_OL_CNT veces (una vez por artículo en la orden).

Comentario 1: No es necesario que l orden de los artículos mostrado en la pantalla de la transacción Order-Status coincida con el orden de introducción de los artículos en la correspondiente pantalla de la transacción New-Order.

Comentario 2: Si OL_DELIVERY_D es un valor nulo (la orden no ha sido repartida) se mostrará un valor de fecha nula específica de la implementación (ej: espacios, 99-99-9999, etc). La representación de fecha nula debe ser constante durante todo el test.

2.7 Transacción Delivery

La transacción Delivery consiste en procesar una tanda de 10 New-Orders (aún sin repartir). Cada orden se procesa como una única transacción de base de datos de lectura/escritura. El número de órdenes que se repartan en grupo dentro de la misma transacción de base de datos es específico de cada implementación. La transacción comercial compuesta de una o más (hasta 10) transacciones de base de datos tiene una frecuencia de ejecución baja y unos requisitos de tiempo de respuesta poco estrictos.

Se espera que la transacción Delivery se ejecute en modo aplazado mediante un mecanismo de encolado en vez de forma interactiva, con una respuesta al terminal que indique que la transacción ha terminado. El resultado de la ejecución aplazada se escribe en un fichero de resutados.

2.7.1 Generación de datos de entrada

2.7.1.1 Dado un terminal, el número de almacén local (W_ID) es constante durante todo el intervalo de medida.

2.7.1.2 El número de transportista (O_CARRIER_ID) se selecciona aleatoriamente entre [1..10].

2.7.1.3 La fecha de reparto (OL_DELIVERY_DATE) se genera en el SUT a partir de la fecha actual del sistema.

2.7.2 Ejecución Aplazada

2.7.2.1 A diferencia de las otras transacciones en este benchmark, la transacción de Reparto debe ejecutarse en modo aplazado. Este modo de ejecución consiste principalmente en el encolado de la transacción para aplazar su ejecución, devolviendo el control al terminal de origen independientemente de la finalización de la transacción, y escribiendo la información sobre ejecución en un fichero de resultados.

2.7.2.2 La ejecución aplazada de la transacción de Reparto debe incluir los siguientes pasos:

1. La transacción comercial se encola tras introducir el ultimo carácter en la entrada.
2. La ejecución aplazada de la transacción comercial debe seguir el perfil de transacción descrito en la Cláusula 2.7.4 con los datos de entrada definidos en la Cláusula 2.7.1 introducidos a través de la pantalla de E/S y comunicados a la cola de ejecución aplazada.
3. Al menos el 90 % de las transacciones comerciales deben completarse dentro de un plazo de 80 segundos tras haberse encolado.
4. En lo referente a la finalización de la transacción comercial, se deberá almacenar la siguiente información en el fichero de resultados:
    - La hora de encolado de la transacción.

- El número de almacén (W_ID) y el número de transportista (O_CARRIER_ID) asociado con la transacción comercial.
- El número de distrito (D_ID) y el número de orden para cada orden repartida por la transacción comercial.
- La hora de finalización de la transacción.

2.7.2.3 El único propósito del fichero de resultados asociado a la ejecución de la transacción DELIVERY es grabar información sobre la transacción y no es relevante para la realización de la función comercial. El fichero de resultados debe mantener las siguientes reglas:

1. Todos los eventos han de haber finalizado antes de que la información relacionada con ellos sea registrada (ej: el registro de un número de distrito o de una orden debe realizarse después de que la transacción de base de datos haya sido validada)
2. No se requiere ninguna propiedad ACID si el fichero sólo se usa para propósitos de benchmark.
3. Durante el intervalo de medida el fichero de resultados debe estar localizado tanto en un medio duradero (ver Cláusula 3.5.1) como en la memoria interna del SUT. En este último caso el fichero de resultados debe ser transferido a un medio duradero después del último intervalo de medida de la ejecución del test.

2.7.3 El Terminal de E/S.

2.7.3.1 Para cada transacción el terminal originado debe mostrar la siguiente pantalla de entrada/salida con todos los campos de entrada/salida limpiados (tanto con espacios como con ceros), excepto el campo de almacén que no cambia y que debe mostrar el valor fijo de W_ID asociado con el terminal.

```
          1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
 1                                Delivery
 2 Warehouse: 9999
 3
 4 Carrier Number: 99
 5
 6 Execution Status: XXXXXXXXXXXXXXXXXXXXXXXX
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

2.7.3.2 El usuario emulado debe introducir en el campo apropiado de la pantalla se E/S, los datos requeridos los cuales se reparten en un único campo: O_CARRIER_ID.

2.7.3.3 El terminal emulado debe mostrar, en el campo apropiado de la pantalla de E/S, todos los datos de entrada y los datos de salida que resulten del encolamiento de la transacción.

2.7.4 El Perfil de transacción

2.7.4.1 La ejecución aplazada del de la transacción de Reparto entrega una orden pendiente (media de 10 artículos por orden) para cada uno de los 10 distritos del almacén seleccionado usando una o más (hasta 10) transacciones de base de datos. A continuación se detallan las pasos realizados en esta transacción.

2.7.4.2 Para un número de almacén dado (W_ID), para cada uno de los 10 distritos (D_W_ID, D_ID) dentro de ese almacén, y para un número de transportista dado (O_CARRIER_ID):

- Los datos de entrada son recuperados de la cola de ejecución aplazada.
- Se comienza la transacción comercial a no ser que ya esté activa al haber sido comenzada para el reparto de una orden previa.
- Se selecciona la fila de la tabla NEW_ORDER que coincide con NO_W_ID (igual W_ID) y NO_D_ID (igual a D_ID) y con el valor más bajo de NO_O_ID. Esta es la orden más antigua sin repartir para ese distrito. Se recupera el número de orden NO_O_ID. Si no se encuentra ninguna fila coincidente, se salta el reparto de una orden de ese distrito, reanudándose el reparto para los distritos restantes del almacén seleccionado. Se debe informar del caso que esta condición se cumpla en más del 1 %, o en más de una vez (lo que sea mas grande) del número total de transacciones comerciales Delivery. El fichero de resultados debe organizarse de tal forma que se pueda determinar el porcentaje de repartos y distritos saltados.
- Se borra la fila seleccionada en la tabla NEW_ORDER.
- Se selecciona la fila de la tabla ORDER que coincide con O_W_ID (igual a D_ID), con O_D_ID (igual a D_ID), y con O_ID (igual a NO_O_ID). Se recupera el número de cliente O_C_ID, y se actualiza O_CARRIER_ID.
- Se seleccionan todas las filas de la tabla ORDER_LINE que coincidan con O_W_ID (igual a W_ID), con O_D_ID (igual a D_ID), y con OL_O_ID (igual a O_ID). Se actualizan todas las fechas de reparto, O_DELLIVERY_DATE, con la hora actual del sistema devuelta por el sistema operativo, y se recupera la suma de todos los OL_AMOUNT.
- Se selecciona la fila de la tabla CUSTOMER que coincida con C_W_ID (igual a W_ID), con C_D_ID (igual a D_ID) y con C_ID (igual a O_C_ID), y se incrementa C_BALANCE con la suma de todas las cantidades de las líneas de artículo (OL_AMOUNT) recuperadas previamente. C_DELIVERY se incrementa en una unidad.
- Se confirma (COMMIT) la transacción de base de datos, en el caso de no se repartan más ordenes dentro de esta transacción de base de datos.

- Se almacena la información de la transacción Delivery en el fichero de resultados.

2.8 Transacción Stock-Level

La transacción Stock-Level determina el número de artículos vendidos recientemente que tienen un nivel de stock por debajo de un umbral especificado. Representa una transacción de base de datos pesada, de sólo lectura con una frecuencia de ejecución baja, y con unos requisitos de tiempo de respuesta y de consistencia poco estrictos.

2.8.1 Generación de datos de entrada

2.8.1.1 Cada terminal utiliza un único valor de W_ID y D_ID, constantes durante toda la medida.

2.8.1.2 El umbral de cantidad mínima en Stock se selecciona aleatoriamente entre [10..20].

2.8.2 Perfil de transacción

2.8.2.1 La comprobación del nivel de stock de los artículos de las últimas 20 órdenes se realiza en una o más transacciones de base de datos mediante los siguientes pasos:

2.8.2.2 Dado un número de almacén (W_ID), un número de distrito (D_W_ID, D_ID), y un nivel umbral de stock dados:

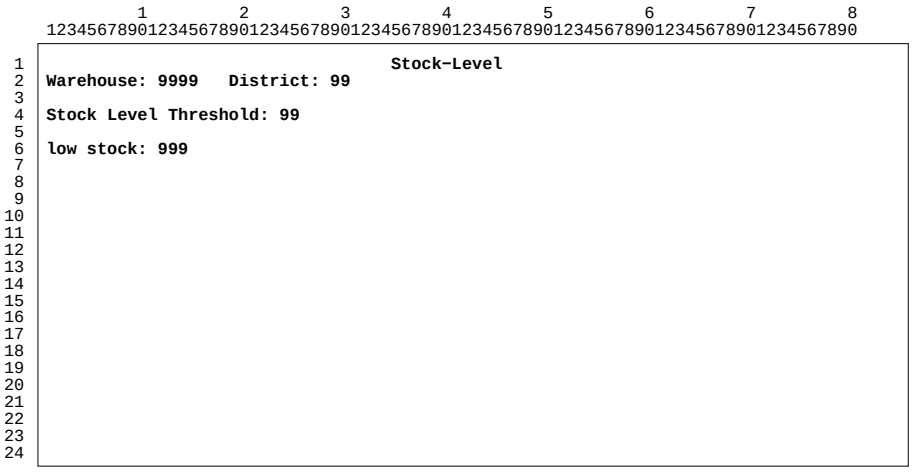
- Los datos de entrada se comunican SUT (ver Cláusula 2.8.3.2).
- Se comienza una transacción de base de datos.
- Se selecciona la fila de la tabla DISTRICT que coincide con D_W_ID, y se recupera D_NEXT_O_ID.
- Se seleccionan todas las filas de la tabla ORDER_LINE que coincidan con OL_W_ID (igual a W_ID), OL_D_ID (igual a D_ID) y con OL_O_ID (menor que D_NEXT_O_ID y mayor o igual que D_NEXT_O_ID menos 20. Esos son los artículos correspondientes a las últimas 20 órdenes de ese distrito.
- Se recuentan todas las filas de la tabla de STOCK que coincidan con S_I_ID (igual a OL_I_ID) y con S_W_ID (igual a W_ID) de la lista de artículos con numero de articulo distinto y que tengan un valor de S_QUANTITY menor que el umbral. El valor resultante corresponde al campo 'low_stock'.

  Comentario: Los artículos comunes en las 20 órdenes seleccionadas, sólo se les debe considerar una ver.
- Se confirma (COMMIT) la transacción de base de datos.
- Los datos de salida se comunican al terminal (ver Cláusula 2.8.2.3).

2.8.2.3 No es necesaria la seriabilidad y las lecturas repetidas para la transacción comercial Stock-Level. Todas las lecturas de datos deben confirmarse (COMMIT) después de los datos confirmados inmediatamente antes del momento de inicio de la transacción. Se deben cumplir todas las demás propiedades ACID.

2.8.3 Terminal E/S

2.8.3.1 En cada transacción el terminal de origen debe mostrar la siguiente pantalla de entrada/salida con todos los campos de estrada limpios (bien utilizando espacios o ceros) exceptuando los campos de almacén y distrito que no cambian y que deben mostrar los valores fijos de W_ID y D_ID asociados con el terminal.



2.8.3.2 El usuario emulado debe introducir, en el campo apropiado de la pantalla de E/S, la entrada necesaria en el campo 'threshold'.

2.8.3.3 EL terminal emulado debe mostrar, en los campos apropiados de la pantalla de E/S, todos los datos de entrada y salida resultantes de la ejecución de la transacción. Deben mostrarse los siguientes campos: W_ID, D_ID, 'threshold', y 'low_stock'.

Cláusula 3: Propiedades de sistema y de transacción

3.1 Las propiedades ACID

El objetivo de esta sección es describir de forma informal las propiedades ACID y especificar una serie de tests que deben realizarse para demostrar que se cumplen estas propiedades.

3.1.1 El sistema puesto a prueba debe mantener las propiedades ACID (Atomicidad, Consistencia, Aislamiento, y Durabilidad) de los sistemas de procesamiento de transacciones durante la ejecución de este benchmark. La única excepción a esta norma es permitir lecturas no repetibles para la transacción de Nivel de Stock (ver Cláusula 2.8.2.3).

3.1.2 No hay serie finita de tests que pueda probar el absoluto cumplimento de las propiedades ACID. El superar los test especificados es una condición necesaria, pero no suficiente para el cumplimiento de las propiedades ACID. Sin embargo, para simplificar el informe, sólo se necesita realizar los tests especificados e incluir los resultados en el Informe Completo de Especificaciones del benchmark.

3.1.3 Se deben habilitar todos los mecanismos necesarios para asegurar el completo cumplimiento de las propiedades ACID, tanto en el periodo de test como el periodo de 8 horas de estado estable. Por ejemplo, si el sistema bajo test utiliza bitácoras de deshacer, debe habilitarse el registro para todas las transacciones incluyendo aquellas que no contemplan la cancelación (ROLLBACK) en su perfil de transacción. Cuando se implemente este benchmark en un sistema distribuido, deben realizarse los test para verificar que las transacciones locales y remotas, incluyendo las transacciones que son procesadas en dos o mas nodos, satisfacen las propiedades ACID (ver en las cláusulas 2.4.1.7, 2.4.1.8, 2.5.1.5, y 2.5.1.6 la definición de transacción local y remota).

3.1.4 Aunque los test ACID no ejecutan todos los tipos de transacciones del TPC- C, todas las transacciones TPC-C deben satisfacer las propiedades ACID.

3.1.5 El patrocinador del test que presente los resultados TPC puede realizar los test ACID en cualquier sistema para el que se especifiquen los resultados, siempre y cuando se use el mismo software ejecutable (ej: sistema operativo, gestor de datos, programas de transacción). Por ejemplo, esta cláusula se aplicaría cuando los resultados se presentan para múltiples sistemas de una misma línea de productos. No obstante, el test de durabilidad descrito en las cláusulas 3.5.3.2 y 3.5.3.3 debe ejecutarse en todos los sistemas medidos. Todos los Informes Completos de Especificaciones deben identificar los sistemas utilizados para verificar los requisitos ACID además de identificar los detalles completos de la realización de los test ACID y de los resultados obtenidos.

3.2 Requisitos de Atomicidad

3.2.1 Definición de propiedad de Atomicidad

El sistema bajo test debe garantizar que todas las transacciones de base de datos son atómicas: el sistema por un lado realizará todas las operaciones individuales sobre los datos, y por otro asegurará que los datos no se vean afectados por ninguna operación

completada de forma parcial.

3.2.2 Test de Atomicidad

3.2.2.1 Realizar una transacción Payment utilizando un almacén, distrito, y numero de cliente seleccionados aleatoriamente, y verificar que los registros de las tablas CUSTOMER, DISTRIT y WAREHOUSE han cambiado apropiadamente.

3.2.2.2 Realizar una transacción Payment utilizando un almacén, distrito, y numero de cliente seleccionados aleatoriamente, y sustituir el COMMIT de la transacción por un ROLLBACK. Verificar que los registros de las tablas CUSTOMER, DISTRIT y WAREHOUSE no han cambiado.

3.3 Requisitos de Consistencia

3.3.1 Definición de propiedad de consistencia

La consistencia es la propiedad de la aplicación que necesita que la ejecución de cualquier transacción de base de datos haga pasar a la base de datos de una estado consistente a otro, asumiendo que inicialmente la base de datos se encontraba en estado consistente.

3.3.2 Condiciones de consistencia

En las siguientes cláusulas se definen doce condiciones de consistencia para especificar el nivel de consistencia de la base de datos tras la mezcla de transacciones TPC-C. Una base de datos, cuando se puebla de acuerdo a la Cláusula 4.3, debe cumplir todas estas condiciones para ser consistente. Si los datos se duplican, todas las copias deben cumplir dichas condiciones. De las doce condiciones, se necesitan únicamente las cuatro primeras para demostrar explícitamente que se cumplen las condiciones de consistencia.

3.3.2.1 Condición de Consistencia 1.

Las entradas en la tablas WAREHOUSE y DISTRIT deben satisfacer la relación:

W_YTD=sum(D_YTD)

para cada almacén definido por (W_ID = D_W_ID).

3.3.2.2 Condición de Consistencia 2

Las entradas de las tablas DISTRIT, ORDER y NEW_ORDER deben satisfacer la relación:

D_NEXT_O_ID-1 = max(O_ID) = max(NO_O_ID)

para cada distrito definido por (D_W_ID = O_W_ID = NO_W_ID) y (D_ID = O_D_ID = NO_D_ID). Esta condición no se aplica a la tabla NEW_ORDER para los distritos que no tienen Nuevas Órdenes pendientes.

3.3.2.3 Condición de Consistencia 3

Las entradas de las tablas en la tabla NEW_ORDER deben satisfacer la relación :

Max(NO_O_ID)-min(NO_O_ID)+1 = [número de filas en la tabla NEW_ORDER para este distrito]

para cada distrito definido por NO_W_ID y NO_D_ID. Esta condición no se aplica a los distritos que no tengan nuevas órdenes pendientes (el número de filas es cero)

3.3.2.4 Condición de consistencia 4

Las entradas de las tablas ORDER y ORDER_LINE deben satisfacer la relación:

Sum(O_OL_CNT) = [número de filas en la tabla ORDER_LINE para ese distrito]

para cada distrito definido por O_W_ID = OL_W_ID y O_D_ID = OL_D_ID.

3.3.2.5 Condición de consistencia 5.

Para cualquier fila en la tabla ORDER, O_CARRIER_ID contiene un valor nulo si y sólo si hay una fila correspondiente en la tabla de NEW_ORDER definida por (O_W_ID, O_D_ID, O_ID) = (OL_W_ID, OL_D_ID, OL_O_ID).

3.3.2.6 Condición de consistencia 6

Para cualquier fila en la tabla ORDER, O_OL_CNT debe contener el número de filas de la tabla ORDER_LINE para la orden correspondiente definida por (O_W_ID, O_D_ID, O_ID) = (OL_W_ID, OLD_ID, OL_O_ID).

3.3.2.7 Condición de Consistencia 7.

Para cualquier fila en la tabla ORDER_LINE, OL_DELIVERY_D contiene un valor nulo si y sólo si en la correspondiente fila en la tabla ORDEN definida por (O_W_ID, O_D_ID, O_ID) = (OL_W_ID, OL_D_ID, OL_O_ID), O_CARRIER_ID contiene un valor nulo.

3.3.2.8 Condición de consistencia 8.

Las entradas en las tablas de WAREHOUSE y HISTORY deben satisfacer la relación:

W_YTD = sum(H_AMOUNT)

para cada almacén definido por (W_ID = H_W_ID).

3.3.2.9 Condición de consistencia 9.

Las entradas en las tablas DISTRIT Y HISTORY deben satisfacer la relación:

D_YTD = sum(H_AMOUNT)

Para cada distrito definido por (D_W_ID, D_ID) = (H_W_ID, H_D_ID).

3.3.2.10 Condición de Consistencia 10.

Las entradas en las tablas CUSTOMER, HISTORY, ORDER y ORDER_LINE deben satisfacer:

C_BALANCE = sum(OL_AMOUNT) - sum(H_AMOUNT)

Donde :

H_AMUONT se selecciona por (C_W_ID, C_D_ID, C_ID) = (H_C_W_ID, H_C_D_ID, H_C_ID)

y OL_AMOUNT es seleccionado por:

(OL_W_ID, OL_D_ID, OL_O_ID) = (O_W_ID, O_D_ID, O_ID) y

(O_W_ID, O_D_ID, O_C_ID) = (C_W_ID, C_D_ID, C_ID) y

(OL_DELIVERY_D no es un valor nulo)

3.3.2.11 Condición de Consistencia 12.

Las estradas en la tabla CUSTOMER y ORDER_LINE deben satisfacer la relación:

C_BALANCE + C_YTD_PAYMENT = sum(OL_AMOUNT)

para cualquier cliente seleccionado aleatoriamente donde OL_DELIVERY_D no contenga una fecha nula.

3.3.3 Tests de Consistencia.

3.3.3.1 Comprobar que inicialmente la base de datos es consistente verificando que se cumplen las cuatro primeras condiciones de consistencia descritas anteriormente. Han de describirse con suficiente detalle los pasos seguidos en la realización de este test para que puedan ser repetibles de forma independiente.

3.3.3.2 Inmediatamente después de realizar los procesos de verificación definidos en la Cláusula 3.3.3.1, realizar lo siguiente:

1. 1. Utilizar el mecanismo estándar de control para enviar transacciones al SUT. La tasa de transacciones debe estar dentro del 10 % de la tasa de tpmC presentada y cumplir todos los requisitos del intervalo de medida (ver Cláusula 5.5), incluyendo el requisito de que el intervalo contenga al menos un check-point (ver Cláusula 5.5.2.2). Se debe ejecutar el SUT a esta tasa durante al menos 5 minutos.

2. 2. Interrumpir el envío transacciones al SUT y repetir los pasos de verificación descritos en la Cláusula 3.3.3.1. La base de datos debe continuar en estado consistente después de la ejecución de las transacciones. Sólo se necesita verificar las

condiciones de consistencia en las filas añadidas a las tablas ORDER y ORDER-LINE tras la verificación anterior.

3.4 Requisitos de Aislamiento

3.4.1 Definición de propiedad de Aislamiento

El aislamiento puede definirse por medio de los fenómenos que pueden suceder durante la ejecución simultanea de varias transacciones de base de datos. Es posible que se den los siguientes fenómenos:

P0. Escritura sucia: La transacción de base de datos T1 lee un elemento y lo modifica. La transacción de base de datos T2 seguidamente modifica o borra ese elemento, y realiza un COMMIT. Si T1 intentase releer dicho elemento puede obtener el valor modificado de T2 o encontrarse con que el elemento se ha borrado.

P1. Lectura sucia: La transacción de base de datos T1 modifica un elemento. La transacción de base de datos T2 seguidamente lee el elemento antes de que T1 realice un COMMIT. Si T1 realizara un ROLLBACK, T2 leería un valor no confirmado que por tanto, nunca ha existido.

P2. Lectura no repetible: La transacción de base de datos T1 lee un dato. Seguidamente la transacción de base de datos T2 modifica ese dato y realiza un COMMIT. Si T1 intentara releer el dato podría obtener el valor modificado o encontrarse con que ese dato se ha borrado.

P3. Fantasma: La transacción de base de datos T1 lee un conjunto de valores N que satisfacen alguna condición de búsqueda. La transacción de base de datos T2 ejecuta, entonces, sentencias que generan uno o mas elementos de datos que satisfacen la condición de búsqueda utilizada por la transacción T1. Si la transacción T1 intentara repetir la lectura inicial con la misma condición de búsqueda, obtendría un conjunto distinto de valores.

Las transacciones anteriores T1 y T2 deben ejecutarse por completo o no ejecutarse.

La siguiente tabla define cuatro niveles de aislamiento con respecto a los fenómenos P0, P1, P2, y P3.

Nivel de Aislamiento	P0	P1	P2	P3
0	No Posible	Posible	Posible	Posible
1	No Posible	No Posible	Posible	Posible
2	No Posible	No Posible	No Posible	Posible
3	No Posible	No Posible	No Posible	No Posible

Se definen los siguientes términos:

T1= Transacción New-Order.

T2= Transacción Payment.

T3= Transacción Delivery.

T4= Transacción Order-Status.

T5= Transacción Stock-Level.

Tn= cualquier transacción arbitraria.

Aunque sea arbitraria, la transacción Tn no puede hacer escrituras sucias.

La siguiente tabla define los requisitos de aislamiento que deben cumplir las transacciones TPC-C.

Cond. nº	Para las transacciones en el conjunto:	los fenomenos:	no deben ser vistos por la transaccion:	Descripcion textual:
1.	{Ti, Tj} 1=i,j=4	P0, P1, P2, P3	Ti	Nivel 3 de aislamiento entre las transacciones New Order, Payment, Delivery y Order−Status.
2.	{Ti,Tn} 1 = i = 4	P0, P1, P2	Ti	Nivel 2 de aislamiento para las transacciones New Order, Payment, Delivery y Order−Status en relacion a una transaccion arbitraria
3.	{Ti, T5} 1 = i = n	P0, P1	T5	Nivel 1 de aislamiento para las transacciones Stock−Level en relacion a las transacciones TPC−C y cualquier transaccion arbitraria.

Se deben habilitar los suficientes mecanismos, tanto a nivel de sistema como a nivel de aplicación para asegurar que se obtiene el aislamiento definido anteriormente.

3.4.2 Test de aislamiento.

Para los esquemas de bloqueo convencionales, se debe probar el aislamiento como se describe a continuación. Los sistemas que implementen otros esquemas de aislamiento pueden requerir técnicas diferentes de comprobación. Es responsabilidad del patrocinador del test la descripción de esas técnicas y de los tests correspondientes. Si se usa un esquema de aislamiento distinto del convencional está permitido implementar esos test de manera diferente, especificando completamente los detalles. (Se muestran ejemplos de técnicas diferentes de comprobación en el Test de aislamiento 7, Cláusula 3.4.2.7).

3.4.2.1 Test de Aislamiento 1.

Este test demuestra el aislamiento en los conflictos de lectura-escritura entre las transacciones de Order-Status y New-Order. Se realizan los siguientes pasos:

1. Comenzar una transacción New-Order T1.
2. Parar la transacción T1 inmediatamente antes del COMMIT.
3. Comenzar una transacción Order-Status utilizando el mismo cliente que en T1. La transacción T2 intenta leer el dato que la orden T1 ha creado.
4. Verificar que la transacción T2 espera.
5. Permitir que la transacción T1 se complete. T2 debería ahora completarse.
6. Verificar que los resultados de T2 coinciden con los datos introducidos por T1.

3.4.2.2 Test de aislamiento 2.

Este test demuestra el aislamiento en los conflictos de lectura-escritura entre las transacciones Order-Status y New-Order, cuando se cancela (ROLLBACK) la transacción New-Order. Se realizan los siguientes pasos:

1. Realizar una transacción Order-Status T0 para un cliente. Dejar que T0 se complete.
2. Comenzar una transacción New-Order T1 para el mismo cliente usado en T0.
3. Parar la transacción T1 inmediatamente antes del COMMIT.
4. Comenzar una transacción Order-Status T2 utilizando el mismo cliente que en T0. La transacción T2 intenta leer los datos que la transacción T1 ha creado.
5. Verificar que la transacción T2 espera.
6. Cancelar la transacción T1. T2 debería ahora completarse.
7. Verificar que los datos devueltos por T2 coinciden con los de T0.

3.4.2.3 Test de aislamiento 3.

Este test demuestra el aislamiento en los conflictos de escritura-escritura entre dos transacciones New-Order. Se realizan los siguientes pasos:

1. Comenzar una transacción New-Order T1.
2. Parar la transacción T1 inmediatamente antes del COMMIT.
3. Comenzar la transacción New-Order T2 para el mismo cliente que T1.
4. Verificar que la nueva transacción T2 espera.
5. Permitir que la transacción T1 se complete. T2 debería ahora completarse.

6. Verificar que el número de orden devuelto por T2 es una unidad mayor que el número de orden para T1. Verificar que el valor de D_NEXT_O_ID refleja el resultado tanto de T1 como de T2; es decir, que se ha incrementado en dos unidades y es una unidad mayor que el número de orden de T2.

3.4.2.4 Test de Aislamiento 4.

Este test demuestra el aislamiento en los conflictos de escritura-escritura entre dos transacciones New-Order cuando se cancela una de ellas. Se realizan los siguientes pasos:

1. Comenzar una transacción New-Order T1 que contenga un número de articulo no válido.
2. Parar la transacción T1 inmediatamente antes de cancelar (ROLLBACK).
3. Comenzar otra transacción New-Order T2 para el mismo cliente que T1.
4. Verificar que la transacción T2 espera.
5. Permitir que la transacción T1 se complete. T2 debería ahora completarse.
6. Verificar que el número de orden devuelto por T2 es una unidad que el anterior número de orden. Verificar que el valor de D_NEXT_O_ID refleja sólo el resultado de T2, es decir que se ha incrementado en una unidad y que es una unidad mayor que el número de orden de T2.

3.4.2.5 Test de Aislamiento 5.

Este test demuestra el aislamiento en los conflictos de escritura-escritura entre las transacciones Payment y Delivery. Se realizan los siguientes pasos:

1. Comenzar una transacción Delivery T1.
2. Parar la transacción T1 antes del COMMIT.
3. Comenzar una transacción Payment utilizando el mismo cliente que una de las Nuevas Órdenes repartidas por T1.
4. Verificar que la transacción T2 espera.
5. Permitir que T1 se complete. T2 debería ahora completarse.
6. Verificar que C_BALANCE refleja los resultados tanto de T1 como de T2.

Comentario: Si la transacción comercial Delivery se realiza utilizando múltiples transacciones de base de datos, la transacción T1 del punto 6 anterior, puede ser una de esas transacciones de base de datos.

3.4.2.6 Test de aislamiento 6.

Este test demuestra el aislamiento en los conflictos de escritura-escritura entre las transacciones Payment y Delivery cuando se cancela (ROLLBACK) la transacción Delivery. Se realizan los siguientes pasos:

1. Comenzar una transacción Delivery T1.
2. Parar la transacción T1 inmediatamente antes del COMMIT.
3. Comenzar una transacción Payment T2 utilizando el mismo cliente que una de las ordenes que han sido repartidas por T1.
4. Verificar que la transacción T2 espera.
5. Cancelar la transacción T1. T2 debería ahora completarse.
6. Verificar que C_BALANCE refleja el resultado de sólo la transacción T2.

3.4.2.7 Test de Aislamiento 7.

Este test demuestra las lecturas repetibles para una transacción New-Order mientras una transacción interactiva actualiza el precio de un articulo. Dados dos números aleatorios de artículo x e y, se realizan los siguientes pasos:

1. Comenzar una transacción T1. Consultar I_PRICE para los artículos x e y. Confirmar (COMMIT) la transacción T1.
2. Comenzar la transacción New-Order T2 utilizando un grupo de artículos que incluya al artículo x dos veces y una al artículo y.
3. Parar la transacción T2 después de consultar el precio del artículo x por primera vez e inmediatamente antes de consultar el precio del artículo 'y' y del artículo 'x' por segunda vez.
4. Comenzar una transacción T3. Incrementar el precio de los artículos 'x' e 'y' en un 10 %.

   Caso A: Si la transacción T3 se para:

   a) Continuar la transacción T2 y verificar que el precio de los artículos 'x' (por segunda vez) e 'y' coincide con los valores leídos por la transacción T1. Confirmar (COMMIT) la transacción T2.

   b) La transacción T3 debería completar y confirmarse (COMMIT).

   c) Comenzar una transacción T4. Consultar I_PRICE para los artículos 'x' e 'y'. Confirmar la transacción T4.

   d) Verificar que los precios leídos por la transacción T4 coinciden con los valores introducidos por T3.

   Caso B: si la transacción T3 no se para y se cancela la transacción T2.

   a) La transacción T3 ha completado y se ha confirmado (COMMIT).

b) Continuar la transacción T2 y verificar que el administrador de datos fuerza su cancelación.

c) Comenzar la transacción T4. Consultar I_PRICE para los artículos 'x' e 'y'. Confirmar (COMMIT) la transacción T4.

d) Verificar que el precio leído por la transacción T4 coincide con el valor introducido por la transacción T3.

Caso C: si se cancela la transacción T3.

a) Verificar que el administrador de datos fuerza la cancelación de la transacción T3.

b) Continuar la transacción T2 y verificar que el precio del artículo 'x' (por segunda vez) e 'y' y coincide con los valores leídos por la transacción T1. Confirmar (COMMIT) la transacción T2.

c) Comenzar la transacción T4. Consultar I_PRICE para los artículos 'x' e 'y'. Confirmar (COMMIT) la transacción T4.

d) Verificar que los precios leídos por la transacción T4 coincide con los valores leídos por las transacciones T1 y T2.

Caso D: Si la transacción T3 no se para y no se cancela ninguna transacción:

a) La transacción T3 se ha completado y se ha confirmado (COMMIT).

b) Continuar la transacción T2 y verificar que el precio de los artículos 'x' (por segunda vez) e 'y' coinciden con los valores leídos por la transacción T1. Confirmar (COMMIT) la transacción T2.

c) Comenzar una transacción T4. Consultar I_PRICE para los artículos 'x' e 'y'. Confirmar (COMMIT) la transacción T4.

d) Verificar que los precios leídos por la transacción T4 coinciden con los introducidos por la transacción T3.

Comentario 1: El test se ha ejecutado correctamente si se ha seguido alguno de los pasos A,B,C o D anteriores. El patrocinador del test debe especificar el caso que se ha seguido durante la ejecución del test.

Comentario 2: Si la implementación utiliza duplicados para la tabla ITEM pero todas las transacciones en el test de aislamiento 7 utilizan la misma copia de la tabla ITEM, no es necesario que las actualizaciones de la tabla se realicen en todas las copias. Esta flexibilidad de las propiedades ACID en las tablas duplicadas sólo es válida bajo las condiciones de anteriores y en el contexto del test de aislamiento 7.

Comentario 3: Las transacciones T1, T2 y T4 no se utilizan medir el rendimiento si no que se usan únicamente para el test de aislamiento 7.

3.4.2.8 Test de aislamiento 8.

Este test demuestra el aislamiento para el nivel 3 (fantasma) de protección entre una transacción Delivery y una transacción New-Order. Se realizan los siguientes pasos:

1. Eliminar todas las filas de la tabla NEW_ORDER pertenecientes a un distrito y un almacén seleccionados al azar.
2. Comenzar una transacción Delivery T1 utilizando el almacén seleccionado.
3. Parar T1 inmediatamente después de leer en la tabla NEW_ORDER el distrito seleccionado. No debe encontrarse ninguna fila coincidente.
4. Comenzar una transacción New-Order T2 utilizando el mismo almacén y distrito.

   Caso A: Si la transacción T2 se para:

   a) Continuar la transacción T1 leyendo de la tabla NEW ORDER el distrito seleccionado.
   b) Verificar que todavía no se ha encontrado ninguna fila coincidente.
   c) Completar y confirmar la transacción T1.
   d) La transacción T2 debería ahora terminar.

   Case B: si la transacción T2 no se para:

   a) Completar y confirmar la transacción T2.
   b) Continuar la transacción T1 repitiendo lectura del distrito seleccionado en la tabla NEW_ORDER.
   c) Verificar que todavía no se ha encontrado ninguna fila coincidente.
   d) Completar y confirmar la transacción T1.

Comentario: hay que resaltar que se pueden dar otros casos diferentes a A y B. El objetivo de este test es demostrar que en todos los casos, tras la lectura de la transacción T1, no se encuentran filas en el distrito seleccionado.

3.4.2.9 Test de aislamiento 9

Este test demuestra el aislamiento para el nivel 3 (fantasma) de protección entre una transacción Order-Status y una New-Order. Se realizan los siguientes pasos:

1. Comenzar una transacción Order-Status T1 seleccionando un cliente.
2. Parar inmediatamente T1 tras leer en la tabla ORDER el cliente seleccionado. Se encuentra la orden más reciente de ese cliente.
3. Comenzar una transacción New-Order T2 utilizando el mismo cliente.

   Caso A, si la transacción T2 se para:

   a) Continuar la transacción T1 repitiendo la lectura de la tabla ORDER utilizando el cliente seleccionado.
   b) Verificar que la orden encontrada es la misma que la del paso 3.
   c) Completar y confirmar la transacción T1.

d) La transacción T2 debería ahora completarse.

Caso B, si la transacción T2 no se para:

a) Completar y confirmar la transacción T2.

b) Continuar la transacción T1 repitiendo la lectura de la tabla de orden para el distrito seleccionado.

c) Verificar que la orden encontrada es la misma que en el paso 3.

d) Completar y confirmar la transacción T1.

Comentario: hay que resaltar que se pueden dar otros casos diferentes a A y B. El objetivo de este test es demostrar que en todos los casos T1 repite la lectura de la tabla ORDER para el cliente seleccionado. La orden encontrada debe ser la misma que en el paso 3.

3.5 Requisitos de Durabilidad

El sistema puesto a prueba debe garantizar la durabilidad: la habilidad de mantener los efectos de las transacciones confirmadas (COMMIT) y asegurar la consistencia tras recuperarse de alguno de los fallos que se especifican en la Cláusula 3.5.3.

Comentario: Ningún sistema está protegido completamente frente a todos los tipos de fallos. El conjunto de fallos simples recogidos en la cláusula 3.5.3 se lo considera suficientemente significativo para asegurar la durabilidad.

3.5.1 Definición de medio duradero

Un medio duradero es un medio de almacenamiento de datos que es:

1. Un medio de naturaleza no volátil (ej: disco magnético, disco óptico, etc.) o
2. Un medio volátil que asegurará la transferencia de datos a un medio no volátil automáticamente, antes de que se pierda cualquier dato tras un fallo de la energía externa independientemente de la reaplicación de la energía externa.

Comentario: Un medio duradero puede fallar; este normalmente se protege utilizando un mecanismo de duplicado (Mirroring) o escribiendo las bitácoras en otro medio duradero. La memoria puede considerarse como un medio duradero si puede preservar los datos lo suficiente como para satisfacer el punto 2 indicado anteriormente, por ejemplo, si esta protegida a través de un Sistema de Alimentación Ininterrumpida, y los contenidos de la memoria pueden transferirse a un medio no volátil durante el fallo. Nótese que no se hace ninguna distinción entre la memoria principal y la memoria que realiza el almacenamiento de los datos de forma temporal o permanente en otras partes del sistema.

3.5.2 Definición de la propiedad de Confirmación (COMIITED)

Una transacción se considera confirmada cuando el gestor de transacciones del sistema ha, o bien escrito la bitácora ('log') y o escrito en un medio duradero los datos de las actualizaciones confirmadas asociadas a la transacción.

Comentario 1: Las transacciones pueden confirmarse sin que el usuario reciba notificación de ese hecho, ya que no es necesaria la integridad de mensaje en el TPC-C.

Comentario 2: Aunque el orden de las operaciones en el perfil de transacción es irrelevante, la comunicación de los datos de salida no puede comenzar hasta que la operación confirmación se haya completado con éxito.

3.5.3 Lista de fallos simples

3.5.3.1 Fallos permanentes irrecuperables de algún medio duradero que contenga las tablas de la base de datos del TPC-C o las bitácoras de recuperación de datos.

Comentario: Si se utiliza la memoria principal como un medio duradero, entonces se debe considerar como un elemento de fallos en potencia. Un método para reponerse de un fallo es almacenar una copia de la base de datos junto con la bitácora de deshacer otro medio duradero. Si el medio duradero es la memoria y el mecanismo utilizado es el anterior, entonces el medio que almacene la copia debe alimentarse de forma independiente.

3.5.3.2 Interrupciones instantáneas (caída del sistema/ cuelgue del sistema) durante el proceso que requieren del re-inicio para la recuperación del sistema.

Comentario: Esto implica la parada anormal del sistema que requiere la recarga de sistema operativo por parte del dispositivo de arranque. No implica necesariamente la pérdida de memoria volátil. Cuando el mecanismo de recuperación se basa en el contenido de la memoria volátil previo al fallo, se debe incluir en el presupuesto del sistema el mecanismo utilizado para evitar la pérdida de la memoria volátil (p.e: un SAI). Un mecanismo que proteja de una interrupción instantánea es por ejemplo una bitácora de hacer/deshacer.

3.5.3.3 Fallo de toda o una parte de la memoria.

Comentario: Puede provenir de una perdida de la alimentación externa o de un fallo permanente en la placa de memoria.

3.5.4 Test de Durabilidad

La intención de este test es demostrar que todas las transacciones cuya salida se haya recibido en el terminal o el ETR se han validado a pesar de haberse producido alguno de los fallos descritos anteriormente y que todas las condiciones de consistencia continúen cumpliéndose después de recuperar la base de datos.

Se requiere que el/los test/s por caída del sistema y el/los test/s por pérdida de memoria descritos en las Cláusula 3.5.3.2 y 3.5.3.3 se realicen a plena carga y con la base de datos completamente escalada. Se puede realizar el test/s, para los fallos del medio duradero descritos en la Cláusula 3.5.3.2, en un subconjunto de la configuración

del SUT y de la base de datos. Para el subconjunto del SUT, todos los componentes hardware, tales como procesadores y discos/controladores en la configuración del SUT, deben estar representados en más de 10 % de la configuración o en dos de cada uno de los componentes hardware múltiples. La base de datos debe estar escalada al menos en un 10 % del escalado completo, con un mínimo de dos almacenes. Debe permitirse al Auditor del TPC una excepción a los requisitos de configuración establecidos anteriormente, para reducir la complejidad del benchmark. Cualquier excepción semejante debe estar documentada en la carta de atestado del auditor. Además en este test debe utilizarse el mecanismo estándar de control. El patrocinador del test debe garantizar que a su entender, un test completamente escalado debería también pasar todos los test de durabilidad.

Para cada uno de los fallos definidos en la cláusula 3.5.3 se realizarán los siguientes pasos:

1. Calcular la suma de D_NEXT_O_ID para todas las filas en la tabla DISTRIT para determinar el número total actual de órdenes (cuenta1).
2. Comenzar a presentar transacciones TPC-C. La tasa de transacción debe ser al menos el 10 % de la tasa tpmC presentada y debe cumplir todos los requisitos en cuanto al intervalo de medida (ver Cláusula 5.5), excluyendo el requisito de que el intervalo de medida contenga un çheckpoint". EL SUT debe funcionar a esta tasa al menos durante 5 minutos. En el Driver System, el sponsor del test debe registrar la confirmación y la cancelación de las transacciones de New-Order en un fichero de estado.
3. Causar el fallo seleccionado de la lista de la Cláusula 3.5.3.
4. Arrancar de nuevo el sistema bajo test usando los procedimientos convencionales de recuperación.
5. Comparar los contenidos del fichero de éxito y la tabla ORDER para verificar que todos los registros en el archivo de estado para la transacción New-Order tienen un correspondiente registro en la tabla ORDER y que no existe ningún apunte de transacciones canceladas.

   Repetir el paso 1 para determinar el número total de órdenes (cuenta2). Verificar que cuenta2-cuenta1 es mayor o igual que el número de registros en el fichero de éxito par las transacciones New-Order confirmadas. Si no son iguales, la tabla ORDER debe contener registros adicionales y la diferencia debe ser menor o igual que el número de terminales simulados.

   Comentario: Esta diferencia sólo se debería a las transacciones que validadas en el SUT, no mostraron su resultado en la pantalla de E/S antes del fallo.
6. Verificar la Condición de Consistencia 3 especificada en la Cláusula 3.3.2.3.

3.5.5 Requisitos adicionales

3.5.5.1 Los mecanismos de durabilidad no pueden usar la tabla HISTORY para mantener las propiedades de durabilidad.

Cláusula 4: Escalado y Poblado de la Base de Datos

4.1 Reglas Generales de Escalado

El rendimiento del benchmark TPC-C viene dado por la actividad de los terminales conectados a cada almacén. Para incrementar el rendimiento, se deben configurar más almacenes y terminales asociados. Cada almacén requiere un número determinado de filas de la población de la base de datos, además se necesita el espacio de almacenamiento suficiente para mantener los datos producidos por un periodo de 60 días de actividad definido como "periodo de 60 días". Esta cláusula describe los requisitos de escalado, espacio de almacenamiento y de la población en función del rendimiento presentado por el sistema.

4.1.1 Los requisitos de escalado tiene como objetivo mantener la proporción entre la carga de transacciones presentada al sistema a prueba, la cardinalidad de las tablas a las que acceden las transacciones, el espacio de almacenamiento requerido y el número de terminales que generan la carga de transacciones.

4.1.2 Si se exceden los valores definidos en la Cláusula 4.2, los otros valores deben ser incrementados proporcionalmente para mantener las relaciones entre ellos definidas en la Cláusula 4.2

4.1.3 El rendimiento declarado no debe exceder del máximo permitido por los requisitos de escalado de la cláusula 4.2 y los requisitos entre etapas de la cláusula 5.2 Aunque el rendimiento declarado pueda caer por debajo del máximo permitido por el sistema configurado, el cálculo de la relación precio/rendimiento (ver cláusula 7.1) debe informar sobre el precio del sistema según está configurado realmente. Para evitar el sobreescalado del sistema, el rendimiento declarado no puede caer por debajo de 9 tpmC por cada almacén configurado.

Comentario: El rendimiento máximo se alcanza con transacciones infinitamente rápidas resultantes de tiempos de respuesta nulos y tiempos de espera mínimos. El objetivo de esta cláusula es prevenir la declaración de un rendimiento que exceda ese máximo, cuyo valor se estima que sea 12,86 tpmC por almacén. El tpmC de 9 mencionado anteriormente representa el 70 % del valor calculado para el rendimiento máximo.

4.2 Requisitos de Escalado

4.2.1 La tabla WAREHOUSE es unidad básica de escalado ya que cardinalidad de las otras tablas (excepto la ITEM) están en función del número de almacenes configurados. Este número, por lo tanto, determina la carga aplicada al sistema bajo test de la que deriva el rendimiento declarado.

Comentario 1: La cardinalidad de las tablas HISTORY, NEW-ORDER, ORDER y ORDER- LINE variarán, como es lógico, proporcionalmente a las ejecuciones repetidas de la prueba. Se diseña la población inicial de la base de datos y los perfiles de transacción

Nombre de tabla	Cardinalidad (en filas)
Warehouse	1
District	10
Customer	30k
History [1]	30k
Order	30k
New−Order	9k
Order−Line	300k
Stock	100k
Item [2]	100k

[1] Pequeñas variaciones: dependientes de la ejecucion de la prueba ya que se pueden añadir o eliminar filas como resultado de la realizacion de las transacciones.

[2] Cardinalidad fija: su escala no depende del numero de almacenes.

Figura A.3: Cardinalidad de la población inicial por almacén.

para minimizar el impacto de esta variación en el rendimiento y mantener la repetibilidad entre resultados de pruebas posteriores.

Comentario 2: La cardinalidad de la tabla ITEM es constante y no depende del número de almacenes configurados ya que todos ellos guardan existencias del mismo catalogo de artículos.

4.2.2 Configuración

Los siguientes requisitos de escalado representan la configuración inicial del test descrito en la cláusula 5:

1. Para cada almacén activo en la base de datos, el SUT debe aceptar peticiones de transacción de un grupo de 10 terminales.
2. Para cada tabla que compone la base de datos, la cardinalidad de la población inicial para cada almacén se especifica como indica la figura A.3.
3. Se debe predecir el espacio suficiente para almacenar y mantener los datos generados durante un periodo de 60 días de actividad con una media de 8 horas al día trabajando al rendimiento declarado y que se conoce como "periodo de 60 días". Se debe calcular este espacio de acuerdo con la cláusula 4.2.3, además el gestor de datos debe poderlo usar para almacenar y mantener las filas que pudieran añadirse en las tablas HISTORY, ORDER y ORDER-LINE durante el periodo de 60 días.
4. El incremento mínimo en el escalado de la base de datos y la población de terminales es de un almacén, que está compuesto por una fila en la tabla WAREHOUSE, diez filas en la DISTRICT, sus filas asociadas en las tablas CUSTOMER, HISTORY, ORDER, NEW ORDER y ORDER-LINE, 10 terminales, y el espacio de almacenamiento calculado para el periodo de 60 días.

Comentario: Se permite sobreescalar la base de datos, por ejemplo, configurar un número mayor de almacenes y tablas asociadas (Wc) de los que se accede realmente durante la medida (Wa), siempre y cuando se cumplan las siguientes condiciones:

Siendo:

- Wc = el número de almacenes configurados en la generación de la base de datos.
- Wa = el número de almacenes a los que se ha accedido durante la medida (almacenes activos).
- Wi = el numero de almacenes a los que no se ha accedido durante la medida (almacenes inactivos)

* Se puede demostrar que no se accede a los almacenes inactivos durante la medida. Se puede demostrar este hecho a través de una de las siguientes formas:

1. Se deben eliminar, antes de la medida, las filas en la tabla WAREHOUSE que pertenecen a almacenes inactivos (Wi).
2. Comprobar que la suma de D_NEXT_O_ID para cada uno de los almacenes inactivos no cambia durante el periodo e medida, y que W_YTD para cada uno de los almacenes inactivos no cambia durante el periodo de medida.

    - El rendimiento declarado no puede caer por debajo de 9 tpmC por almacén configurado (Wc- ver cláusula 4.1.3)
    - Los cálculos de espacio para 60 días deben ser realizados basándose en Wc, el número de almacenes configurados al generar la base de datos.

4.2.3 Cálculo del espacio de 60 días

El espacio de almacenamiento requerido en el periodo de 60 días se debe determinar como sigue:

1. Se debe construir la base de datos de la prueba incluyendo la población inicial de la base de datos (ver cláusula 4.3) y todos los índices presentes durante la prueba.
2. Se debe construir la base de datos de la prueba para soportar el rendimiento declarado durante un periodo de ocho horas. Esto excluye realizar en la base de datos cualquier operación que no ocurra durante el intervalo de medida (ver cláusula 5.5).
3. El espacio de almacenamiento total para la base de datos de la prueba debe componerse de lo siguiente:

    Espacio Libre: cualquier espacio asignado a la base de datos de la prueba que esté disponible para un uso futuro. Lo compone todo el espacio de almacenamiento de la base de datos que no se usa para almacenar entidades de la base de datos (ej: una fila, un índice, un metadato) o aquello que el gestor de datos no usa como formato de cabecera.

Espacio Dinámico: Cualquier espacio que se usa para almacenar filas existentes de las tablas dinámicas (ej las tablas HISTORY, ORDER y ORDER- LINE). Lo compone todo el espacio de almacenamiento de la base de datos que se usa para almacenar filas y cabeceras de almacenamiento de filas para las tablas dinámicas. Incluye cualquier dato que se añade a la base de dato como resultado de la inserción de una fila independientemente de todos los índices. No incluye los índices de datos u otras cabeceras como la cabecera de índice, cabecera de página, cabecera de bloque y la cabecera de tabla.

Espacio Estático: cualquier espacio utilizado para almacenar información estática e índices. Lo compone todo el espacio reservado para la base de datos de la prueba que no esté incluido tanto en el Espacio Libre como en el Espacio Dinámico.

4. Dado que se debe configurar el sistema para soportar el rendimiento declarado durante un periodo de ocho horas, la base de datos debe permitir el crecimiento de las tablas dinámicas durante al menos ocho horas sin afectar al rendimiento. Se llama crecimiento diario al espacio libre utilizado para permitir el crecimiento de las tablas dinámicas durante 8 horas al día trabajando al rendimiento declarado. Dado W, el numero de almacenes configurados en el sistema en pruebas, el crecimiento diario debe calcularse como:

   Crecimiento-Diario= (espacio-dinámico / (W * 62,5))*tpmC

   Nota: en la fórmula anterior, se usa 62,5 como un factor de normalización ya que la población de la base de datos inicial para cada almacén mantiene el espacio dinámico necesario para la actividad diaria de 8 horas a 62.5 tpmC.

5. Cualquier espacio que sobrepase el 150 % del crecimiento diario se denomina Extensión diaria, y debe añadirse al espacio dinámico cuando se calcule el almacenamiento necesario para el periodo de 60 días. La extensión diaria se calcula de la siguiente forma:

   Extensión-Diaria = Espacio-Libre - 1.5*Crecimiento-Diario

   Si el cálculo de la extensión diaria resulta negativo, se utilizará un valor nulo para la extensión diaria.

6. El espacio de 60 días se calcula como:

   Espacio de 60 días = Espacio-Estático+60*(Crecimiento-diario+ Extensión-Diaria)

7. Se considera parte del espacio de 60 días al espacio dinámico presente en la base de datos de la prueba.

4.3 Población de la Base de Datos

4.3.1 La prueba descrita en la cláusula 5 precisa de un adecuado escalado en la base de datos. Cada tabla debe contener, antes de la ejecución de la prueba, el número de

filas definidas en la cláusula 4.2.2 (ej la taba New-Order debe contener 9000 filas por almacén)

4.3.2 Definición de términos

4.3.2.1 El término **random** significa seleccionado independientemente y con distribución uniforme entre el rango de valores especificado.

Comentario: Sólo cuando se pueble la base de datos inicial, los números aleatorios se pueden generar seleccionando entradas secuenciales entre al menos 10.000 números aleatorios pregenerados. No se debe usar esta técnica para el campo O_OL_CNT.

4.3.2.2 La notación **random a-string [x .. y]** representa una cadena de caracteres alfanuméricos de longitud aleatoria entre un mínimo valor de x y un máximo valor de y, y media (x+y)/2. La notación **n-random [x..y]** una cadena numérica con las mismas características.

Comentario 1: El conjunto de caracteres usado debe ser capaz de representar un mínimo de 128 caracteres.

Comentario 2: Se puede implementar la generación de esas cadenas concatenando dos cadenas seleccionadas aleatoriamente de dos matrices separadas de cadenas, y donde:

1. Ambas matrices contienen un mínimo de 10 cadenas de caracteres diferentes.
2. La primera matriz contiene cadenas de x caracteres.
3. La segunda matriz contiene cadenas de longitudes uniformemente distribuidas entre 0 y (x-y) caracteres.
4. Ambas matrices pueden contener cadenas que tengan relación con la fila y el atributo (ej: usar un nombre real para C_FIRST) en lugar de caracteres aleatorios, siempre que esto no suponga una mejora en el rendimiento medido.

4.3.2.3 El 'último nombre de cliente' (C_LAST) se genera concatenando tres de las siguientes sílabas de longitud variable:

0	1	2	3	4	5
BAR	OUGTH	ABLE	PRI	PRES	ESE

6	7	8	9
ANTI	CALLY	ATION	EING

Dado un número entre 0 y 999, cada una de las tres cifras determina la sílaba concatenar. Por ejemplo, el numero 371 genera el nombre PRICALLYOUGHT, y el número 40 genera el nombre BARPRESBAR.

4.3.2.4 La notación **unique within [x]** representa un valor único dentro de un conjunto de x valores correlativos. Cuando se pueblan varios grupos de filas del mismo tipo

(p.e. cada distrito tiene su propio grupo de clientes), cada grupo debe usar el mismo conjunto de x valores correlativos.

4.3.2.5 La notación **random within [x..y]** representa un valor aleatorio seleccionado independientemente y con distribución uniforme entre x e y, ambos inclusive, con una media (x+y)/2, con la precisión indicada. Por ejemplo, [0,01 .. 1000,00] tiene 10.000 valores únicos,mientras que [1..100] tiene tan solo 100 únicos valores.

4.3.2.6 La notación **random permutation of [x..y]** representa una secuencia de números de x a y desordenados aleatoriamente. Esto se conoce comúnmente permutación (o selección) sin reemplazo.

4.3.2.7 El código postal del almacén (W_ZIP), el código postal del distrito (D_ZIP) y el código postal del cliente (C_ZIP) deben generarse concatenando una cadena numérica aleatoria de 4 caracteres seguida de una cadena constante '11111'. Esto creará 10.000 códigos "zip"únicos.

Comentario: Con 30.000 clientes por almacén y 10.000 códigos postales disponibles, habrá una media de 3 clientes por almacén con el mismo código postal.

4.3.3 Requisitos para Población de las Tablas.

4.3.3.1 La población inicial de la base de datos debe componerse de:

- 100.000 filas en la tabla ITEM con:

  I_ID unique within [100.000]

  I_IM_ID random within [1..10.000]

  I_NAME random a-string[14..24]

  I_PRICE random within [1,00..100,00]

  I_DATA random a-string[26..50] Para el 10 % de las filas, seleccionadas aleatoriamente, la cadena .ºRIGINAL"debe estar contenida dentro de I_DATA en 8 posiciones consecutivas empezando por una posición elegida aleatoriamente.

- 1 fila en la tabla WAREHOUSE para cada almacén configurado con:

  W_ID unique within [numero_de_almacenes_configurados]

  W_NAME random a-string[6..10]

  W_STREET_1 random a-string[10..20]

  W_STREET_2 random a-string[10..20]

  W_CITY random a-string[10..20]

  W_STATE random a-string de 2 letras

  W_ZIP generado de acuerdo con la cláusula 4.3.2.7

  W_TAX random within [0,0000 .. 0,2000]

W_YTD = 300.000,00

Para cada fila en la tabla WAREHOUSE:

- 100.000 filas en la tabla STOCK con:

  S_I_ID unique within [100.000]
  S_W_ID = W_ID
  S_QUANTITY random within [10..100]
  S_DIST_01 random a-string de 24 letras
  S_DIST_02 random a-string de 24 letras
  S_DIST_03 random a-string de 24 letras
  S_DIST_04 random a-string de 24 letras
  S_DIST_05 random a-string de 24 letras
  S_DIST_06 random a-string de 24 letras
  S_DIST_07 random a-string de 24 letras
  S_DIST_08 random a-string de 24 letras
  S_DIST_09 random a-string de 24 letras
  S_DIST_10 random a-string de 24 letras
  S_YTD = 0
  S_ORDER_CNT = 0
  S_REMOTE_CNT = 0
  S_DATA random a-string[26..50] Para el 10 % de las filas, seleccionadas aleatoriamente, la cadena .ºRIGINAL"debe estar contenida dentro de I_DATA en 8 posiciones consecutivas empezando por una posición elegida aleatoriamente.

- 10 filas en la tabla DISTRICT con:

  D_ID unique within [10]
  D_W_ID = W_ID
  D_NAME random a-string[6..10]
  D_STREET_1 random a-string[10..20]
  D_STREET_2 random a-string[10..20]
  D_CITY random a-string[10..20]
  D_STATE random a-string de 2 letras
  D_ZIP generado de acuerdo con la cláusula 4.3.2.7
  D_TAX random within [0,0000..0,2000]
  D_YTD = 30.000,00
  D_NEXT_O_ID = 3.001

  Para cada fila en la tabla DISTRICT:

  - 3.000 filas en la tabla CUSTOMER con:

    C_ID unique within [3.000]
    C_D_ID = D_ID

C_W_ID = D_W_ID

C_LAST generado de acuerdo con la Cláusula 4.3.2.3 iterando desde el rango de [0..999] para los primeros 1000 clientes, y generando un número aleatorio con distribución no uniforme usando la función NURand(255,0,999) para cada uno de los 2000 clientes restantes. La constante de tiempo de ejecución C (ver cláusula 2.1.6) usada para la población de la base de datos debe elegirse aleatoriamente de forma independiente a la ejecución del test.

C_MIDDLE = "0E"

C_FIRST random a-string[8..16]

C_STREET_1 random a-string[10..20]

C_STREET_2 random a-string[10..20]

C_CITY random a-string[10..20]

C_STATE random a-string de 2 letras

C_ZIP generado de acuerdo a la cláusula 4.3.2.7

C_PHONE random n-string de 16 números

C_SINCE fecha/hora dada por el sistema operativo en el que se ha poblado la tabla CUSTOMER C_CREDIT = "GC"Para el 10 % de las filas, elegido aleatoriamente, C_CREDIT = "BC"

C_CREDIT_LIM = 50.000,00

C_DISCOUNT random within [0,0000..0,5000]

C_BALANCE = -10,0

C_YTD_PAYMENT = 10,0

C_PAYMENT_CNT = 1

C_DELIVERY_CNT = 0

C_DATA random a-string[300..500]

Para cada fila en la tabla CUSTOMER:

- ◇ 1 fila en la tabla HISTORY con:

H_C_ID = C_ID

H_C_D_ID = H_D_ID = D_ID

H_C_W_ID = H_W_ID = W_ID

H_DATE fecha y hora actual

H_AMOUNT = 10,00

H_DATA random a-string[12..24]

- ◦ 3.000 filas en la tabla ORDER con:

O_ID unique within [3.000]

O_C_ID

O_D_ID = D_ID

O_W_ID = W_ID

O_ENTRY_D fecha/hora actual dada por el sistema operativo

O_CARRIER_ID random within [1..10] si O_ID <2.001, nulo en otro caso

O_OL_CNT random within [5..15]

O_ALL_LOCAL = 1

Para cada fila en la tabla ORDER:

- ◇ Un numero de filas en la tabla ORDER-LINE igual a O_OL_CNT, generado de acuerdo a las reglas de la generación de datos de entrada de la transacción New-Order (ver Cláusula 2.4.1) con:

  OL_O_ID = O_ID

  OL_D_ID = D_ID

  OL_W_ID = W_ID

  OL_NUMBER unique within [O_OL_CNT]

  OL_I_ID random within [1..100.000]

  OL_SUPPLY_W_ID = W_ID

  OL_DELIVERY_D = O_ENTRY_D si OL_O_ID <2.201, nulo en otro caso

  OL_QUANTITY = 5

  OL_AMOUNT = 0,00 si OL_O_ID <2.201, random within [0,01..9.999,99] en otro caso.

  OL_DIST_INFO random a-string de 24 letras

- ◦ 900 filas en la tabla NEW-ORDER correspondientes a las ultimas 900 filas en la tabla ORDER para ese distrito (p.e, con NO_O_ID entre 2.101 y 3.000) con:

  NO_O_ID = O_ID

  NO_D_ID = D_ID

  NO_W_ID = W_ID

Comentario: Se permite el cinco por ciento (5 %) de variación de la cardinalidad del objetivo de S_DATA con 'ORIGINAL', I_DATA con 'ORIGINAL', y C_CREDIT con 'BC' para atender a la variación que se da durante la carga inicial de datos de la base de datos.

4.3.3.2 La implementación no puede tomar ventaja del echo de que algunos campos sean poblados inicialmente con un valor fijo. Por ejemplo, el espacio de almacenamiento no debe guardarse definiendo un valor por defecto para el campo C_CREDIT_LIM y almacenar este valor tan solo una vez en la base de datos.

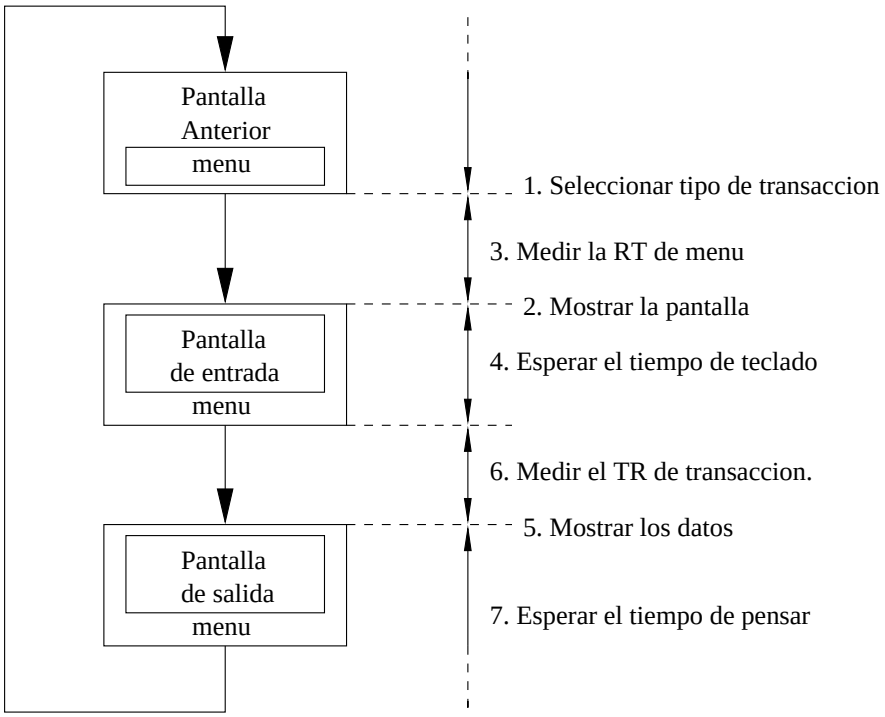


Figura A.4: Ciclo de usuario emulado

Cláusula 5: Medida de Rendimiento y Tiempo de Respuesta

5.1 Definición de términos

5.1.1 Se entiende como intervalo de medida al periodo en estado estable durante la ejecución del benchmarck en el que se está calculando el rendimiento del sistema (ver cláusula 5.5 para más información).

5.1.2 Se entiende como transacción completada a aquella transacción comercial confirmada por el SUT (ver Cláusula 2.1.3) cuyos datos de salida se han mostrado en el emulador de terminal remoto (en el caso de las transacciones New-Order, Payment, Order-Status y Stock-Level) o se han escrito en un fichero de resultados (en el caso de la transacción Delivery), como se especifica en la cláusula 2.1.1.4.

5.2 Pasos de los Usuarios Emulados para las Transacciones

5.2.1 La figura A.4 ilustra el ciclo de ejecución de cada usuario emulado (ver cláusula 5.2.2). La parte activa de la pantalla se representa con texto en negrita:

5.2.2 Cada usuario emulado ejecuta un ciclo compuesto de muestras de pantalla, tiempos de espera y tiempos de respuesta (RTs) como se define a continuación:

1. Selección de un tipo de transacción a partir del menú, de acuerdo con una distribución ponderada (ver cláusula 5.2.3).

2. Espera hasta mostrar la pantalla de entrada/salida.
3. Medición del tiempo de respuesta del menú (ver cláusula 5.3.3).
4. Introducción del número de campos de entrada necesarios (ver cláusula 2) durante el tiempo mínimo de escritura definido (ver cláusula 5.2.5.2).
5. Espera hasta mostrar el número de campos de salida necesarios (ver cláusula 2) en la pantalla de entrada salida.
6. Medición del TR de Transacción (ver cláusula 5.3.4).
7. Espera del tiempo de Pensar mínimo definido (ver cláusula 5.2.5.4) mientras la pantalla de entrada/salida continúa mostrándose.

Al final del Tiempo de Pensar (paso 7) el usuario emulado retorna a seleccionar un tipo de transacción en el menú (paso 1). **Comentario:** No es necesaria ninguna acción por parte del SUT para pasar del paso 7 al paso 1.

5.2.3 A través del menú, los terminales puede solicitar cualquier tipo de transacción (p.e transacción de negocios). Durante el intervalo de medida, la población de terminales debe mantener una mínima mezcla de tipos de transacción como sigue:

Tipo de Transaccion	% minimo de mezcla
New−Order [1]	no disp
Payment	43.0
Order−Status	4.0
Delivery	4.0
Stock−Level	4.0

[1] No hay un minimo para la transaccion New−Order ya que su tasa medida es el rendimiento declarado.

Comentario: el objetivo del porcentaje mínimo de mezcla de transacciones es ejecutar aproximadamente una transacción Payment para cada transacción New-Order y aproximadamente una transacción Order-Status, una transacción Delivery, y una transacción Stock-Level por cada 10 transacciones New-Order.

5.2.4 Regulación de la Mezcla de Transacciones

La selección del tipo de transacción debe realizarse de forma aleatoria siempre y cuando se mantenga el porcentaje mínimo de mezcla de transacciones. Para controlar este porcentaje se puede usar alguna de las técnicas descritas en las cláusulas 5.2.4.1 y 5.2.4.2.

5.2.4.1 Se asocia un peso a cada tipo de transacción. La mezcla se alcanza seleccionando cada transacción nueva aleatoriamente con una distribución ponderada. Se deben satisfacer los siguientes requisitos cuando se use esta técnica:

1. El patrocinador del test elige los pesos reales acordes con los porcentajes mínimos de mezcla de la cláusula 5.2.3.

2. Con el propósito de alcanzar la mezcla de transacciones requerida, el ETR (Emulador de Terminal Remoto) puede ajustar dinámicamente los pesos asociados a cada tipo de transacción durante el intervalo de medida. Se deben limitar los ajustes a una variación del +-5 % del valor inicial.

5.2.4.2 Se asocian una o más cartas de una baraja a cada tipo de transacción del menú. La mezcla se alcanza seleccionando aleatoriamente con distribución uniforme cada transacción, de una baraja que garantice la mezcla de transacciones requerida. Se deben satisfacer los siguientes requisitos cuando se use esta técnica.

1. Cualquier número de terminales puede compartir la misma baraja con mínimo de una baraja por terminal o una baraja para todos los terminales.

2. Una baraja debe estar compuesta por uno o más conjuntos de 23 cartas (ej: 10 cartas New-Order, 10 cartas Payment y una carta cada una de Order-Status, Delivery y Stock-level). El tamaño mínimo de una baraja es un conjunto por terminal que comparta esa baraja. Si se usa más de una baraja, todas las barajas deben tener el mismo tamaño.

   Comentario: Se puede conseguir generar el máximo porcentaje de New-Orders, manteniendo la mezcla necesaria, por ejemplo, compartiendo una baraja de 230 cartas entre 10 terminales, por ejemplo.

3. Todos los terminales deben utilizar la misma técnica para seleccionar las seleccionar transacciones. No se permite ganar rendimiento o lograr una ventaja en la relación precio/rendimiento controlando uno o más terminales de forma diferente al resto de la población de terminales.

5.2.5 Restricciones en cuanto al Tiempo de Espera y Tiempo de Respuesta (TR)

5.2.5.1 El paso de menú es independiente de la transacción. Al menos el 90 % de todas las selecciones del menú deben tener un TR de menú (ver cláusula 5.3.3) menor de 2 segundos.

5.2.5.2 Para cada tipo de transacción, el tiempo de teclado es constante y debe ser de cómo mínimo 18 segundos para la New-Order, 3 segundos para la Payment, y 2 segundos para las Order-Status, Delivery y Stock-level.

5.2.5.3 Al menos el 90 % de todas las transacciones de cada tipo deben tener un TR de transacción (ver cláusula 5.3.4) de menos de 5 segundos en las New-Order, Payment, Order-Status y Delivery, y de 20 segundos en la Stock-level.

Comentario: En el cálculo del tiempo de respuesta, se incluyen también aquellas transacciones New-Order que se cancelan (rollback), según se especifica en la cláusula 2.4.1.4.

5.2.5.4 Para cada tipo de transacción, el tiempo de pensar se elige de forma independiente por medio de una distribución exponencial negativa. El tiempo de pensar Tt se calcula con la siguiente ecuación:

$$Tt = -\log(r) * mTt$$

- Donde: log = logaritmo natural(base e)
  - Tt = tiempo de pensar
  - r = número aleatorio con distribución uniforme entre 0 y 1
  - mTt = tiempo de pensar medio

Cada distribución debe truncarse a 10 el valor de su medio.

5.2.5.5 El comienzo de los tiempos de espera (Tiempo de teclado y de Pensar) debe tomarse después de que el emulador de terminal haya mostrado el último carácter de la salida (ver Cláusula 2.2.2).

5.2.5.6 El tiempo de respuesta del 90 % de las transacciones New-Order, Payment, Order-Status, Stock-Level y de la parte interactiva de la transacción Delivery debe ser mayor o igual que el tiempo de respuesta medio para cada transacción. Este requisito es para el tiempo de respuesta del terminal y no se aplica a la parte aplazada de la transacción Delivery ni al paso del menú.

5.2.5.7 La siguiente tabla resume la mezcla de transacciones, los tiempos de espera y las restricciones de tiempo de respuesta.

Tipo de Transaccion	% minimo de mezcla	Tiempo Minimo de Teclado	Restriccion de Tiempo de Respuesta del 90% de las Transacciones	Media Minima de la Distribucion de Tiempo de Pensar
New−Order	no disp	18 s	5 s	12 s
Payment	43.0	3 s	5 s	12 s
Order−Status	4.0	2 s	5 s	10 s
Delivery [1]	4.0	2 s	5 s	5 s
Stock−Level	4.0	2 s	20 s	5 s

[1] El tiempo de respuesta se refiere a la respuesta del terminal (conocimiento de que la transaccion ha sido encolada), no a la ejecucion de la transaccion en si. Al menos el 90% de las transacciones se deben completar en 80 segundos tras su encolamiento (ver Clausula 2.7.2.2).

Comentario 1: Las restricciones de tiempo de respuesta se eligen para que el rendimiento dependa del tiempo de respuesta de la transacción New-Order. Las restricciones del tiempo de respuesta para las otras transacciones no tienen importancia en este aspecto.

Comentario 2: Los tiempos de escritura para las transacciones se eligen para que sean proporcionales aproximadamente al número de caracteres de la salida.

5.2.5.8 Para cada tipo de transacción, todos los terminales configurados para los sistemas a prueba deben utilizar el mismo criterio en cuanto al tiempo de escritura y al tiempo medio de pensar. Esos tiempos deben cumplir con los requisitos descritos en la Cláusula 5.2.5.7.

5.3 Definición de Tiempo de Respuesta

5.3.1 Cada transacción completada que se envíe al SUT debe cronometrarse individualmente.

5.3.2 Los tiempos de respuesta deben medirse en el ETR. Se define Tiempo de Respuesta (o RT) como:

TR = T2 - T1

Donde T1 y T2 se miden en el ETR y se definen como:

T1: sello de hora tomado antes de que el usuario emulado introduzca el último carácter de la entrada.

T2: sello de hora tomado después de que el terminal emulado reciba el último carácter de la salida.

La resolución de los sellos de hora debe ser de al menos 0.1 segundo.

Comentario: El objetivo del benchmark es medir el tiempo de respuesta como lo experimentara el usuario emulado.

5.3.3 El Tiempo de Respuesta de Menú (TR de Menú) es el tiempo entre el sello de hora tomado antes de que se haya introducido el último carácter de la selección del menú y el sello de hora tomado después de que se haya recibido el último carácter de la pantalla de entrada/salida (incluyendo el borrado de todos los campos de entrada y salida y la muestra de los campos fijos, ver cláusula 2)

Comentario: Los sistemas que no requieren interacción del SUT/ETR para la selección del menú y la muestra de la pantalla pueden asumir un tiempo de respuesta nulo.

5.3.4 El Tiempo de Respuesta de Transacción (TR de Transacción) es el tiempo entre el sello de hora tomado antes de que el ETR haya mandado el último carácter de la salida (ver cláusula 2) y el sello de hora tomado después de que el ETR haya recibido el último carácter de la salida (ver cláusula 2) resultante de una transacción.

Comentario: Si el terminal emulado debe procesar el dato que se introduce o muestra, el tiempo de este proceso debe explicarse y tomarse en el cálculo del TR de Transacción.

5.4 Cálculo de la Tasa de Rendimiento.

La mezcla de transacciones TPC-C representa un ciclo comercial completo. Se compone de múltiples transacciones que introducen nuevas ordenes, consultan el estado de las ordenes existentes, reparten ordenes pendientes, recogen el pago de clientes y revisan el estado de los niveles de stock en los almacenes.

5.4.1 La métrica usada para medir el 'Maximum Qualified Throughput' (MQTh) que es el número de ordenes procesadas por minuto. Es una medida del 'rendimiento comercial' más que una tasa de ejecución de transacciones. Toma en cuenta, implícitamente,

a todas las transacciones de la mezcla ya que su rendimiento individual se controla por los pesos de la selección del menú y por los porcentajes mínimos de mezcla definidos en la Cláusula 5.2.3.

5.4.2 El rendimiento ofrecido "MQTh"es el número de transacciones New-Order ejecutadas por completo (ver Cláusula 5.1.2), dividido por el tiempo transcurrido del intervalo de medida. Las transacciones New-Order canceladas, como especifica la cláusula 2.4.1.1, deben incluirse en el calculo del MQTh.

5.4.3 El nombre de la métrica utilizada para expresar el rendimiento del SUT es 'tpmC'.

5.4.4 El rendimiento declarado debe medirse, no interpolarse o extrapolarse. Debe estar expresado con dos cifras decimales, redondeando la tercera. Por ejemplo, supongamos que se miden 105,548 tmpC en un test con 100 terminales para el cual el 90 % de las transacciones New-Order se han completado en menos de 4.8 segundos y 117,572 tpmC en un test con 110 terminales para el cual el 90 % de las transacciones New-Order se han completado en menos de 5.2 segundos. El rendimiento declarado debe ser 105,54 en vez de un valor interpolado entre 105.548 y 117.572.

5.5 Requisitos del Intervalo de Medida

5.5.1 Estado Estable

5.5.1.1 La prueba debe realizarse en una condición de estado estable que represente el rendimiento real sostenible del SUT.

5.5.1.2 Aunque el intervalo de medida puede ser de tan solo 120 minutos, el SUT debe configurarse para que sea posible el funcionamiento de la prueba a la tasa de rendimiento declarada durante un periodo de al menos ocho horas, manteniéndose por completo las propiedades ACID. Por ejemplo, se debe configurar el dispositivo de almacenamiento para almacenar al menos 8 horas de datos de recuperación si se necesita recuperar el funcionamiento tras un fallo puntual (ver cláusula 3.5.3.1)

Comentario 1: Una configuración que no cumpliría esto, por ejemplo, sería aquella en la que se coloca un fichero de recuperación de tal forma que se alcanza mayor rendimiento durante la parte de medida de la prueba que durante el resto de las ocho horas de la prueba, quizá porque se use un dispositivo dedicado inicialmente pero más tarde se use el espacio de un dispositivo compartido en el periodo completo de ocho horas.

Comentario 2: El estado estable es fácil de definir (ej: rendimiento sostenible) pero difícil de probar. Se requiere que el sponsor de la prueba ponga en conocimiento el método usado para verificar el rendimiento sostenible en estado estable. Se exige al auditor el uso de herramientas de monitorización para ayudar a determinar el estado estable.

Comentario 3: Algunos aspectos de una implementación pueden derivar en variaciones sistemáticas pero de poco valor durante un periodo de 8 horas. El efecto acumulado de estas variaciones puede superar el 2 % del rendimiento declarado. No hay ningún requisito para una ejecución de 8 horas.

5.5.1.3 En el caso de que se use un periodo de *rampa ascendente* para alcanzar el estado estable, se requiere que la base de datos inicial esté adecuadamente escalada al comienzo del periodo rampa. Como en el estado estable, la mezcla de transacciones y los requisitos recogidos en la Cláusula 5.2.5.7 deben cumplirse durante el periodo rampa.

Comentario: el objetivo de esta cláusula es evitar alteraciones significativas del escalado de la base de datos inicial durante el periodo de rampa.

5.5.1.4 No se requiere una medición por separado para demostrar la reproductibilidad.

5.5.1.5 Aunque se permite cierta variación, el ETR no puede ajustarse artificialmente para generar datos de entrada diferentes de los requisitos descritos por las cláusulas 2.4.1, 2.5.1, 2.6.1, 2.7.1 y 2.8.1. Para que sean válidos, los datos de entrada generados durante un intervalo de medida especifico no deben variar más de lo siguiente:

1. Al menos el 0,9 % y como máximo el 1,1 % de las transacciones New-Order deben cancelarse (ROLLBACK) como resultado de un número de artículo no válido.
2. El número medio de order-lines por Order debe estar en el rango del 9,5 al 10,5 y el número de order-lines por order debe ser distribuido uniformemente de 5 a 15 para las transacciones New_order que se solicitan al SUT durante el intervalo de medida.
3. El número de order-lines remotas debe ser de al menos el 0,95 % y como máximo el 1,05 % del numero de order-lines que se rellenan por las transacciones New-Order que se solicitan al SUT durante el intervalo de medida.
4. El número de transacciones Payment remotas debe ser de al menos el 14 % y como máximo del 16 % del número de transacciones Payment que se solicitan al SUT durante el intervalo de medida.
5. El número de selecciones por el último nombre del cliente en la transacción Payment debe ser de al menos el 57 % y como máximo del 63 % del número de transacciones Payment que se solicitan al SUT durante el intervalo de medida.
6. El número de selecciones del cliente por el último nombre del cliente en la transacción Order-Status debe ser de al menos el 57 % y como máximo el 63 % del número de transacciones Order-Status que se solicitan al SUT durante el intervalo de medida.

5.5.1.6 Para que sea válido, el intervalo de medida debe contener como máximo un 1 % y como mínimo una (1), el mayor de ambos casos, de las transacciones Delivery que se rechazan por haber menos ordenes de las necesarias en la tabla New-Order.

5.5.2 Duración.

5.5.2.1 El intervalo de medida debe:

1. Comenzar después de que el sistema alcanza el estado estable.

2. Ser lo suficientemente extenso para generar resultados de rendimiento reproducibles que sean representativos del rendimiento que se alcanzaría durante un periodo sostenido de ocho horas.

3. Permanecer ininterrumpido durante un mínimo de 120 minutos.

5.5.2.2 Algunos sistemas no escriben en un medio duradero los registros/páginas modificados de base de datos en el momento de la modificación, sino que aplazan esta tarea. El proceso por el cual la base de datos escribe los registros/paginas para actualizar la copia durable se denomina CHECKPOINT en este documento.

Se requiere, que en los sistemas que aplazan las escrituras de base de datos en un medio duradero:

1. El tiempo entre los checkpoints (conocido como intervalo de checkpoint (IC)), sea menor o igual que 30 minutos.

   Comentario: Los sistemas de base de datos que se recuperen de interrupciones instantáneas a través de datos de recuperación, no deben usar datos de recuperación que sean anteriores a 30 minutos antes de la interrupción.

2. Todo el trabajo necesario para realizar un checkpoint debe realizarse al menos una vez antes y al menos cuatro veces durante el Intervalo de Medida. Se debe poner en conocimiento el tiempo de comienzo y la duración en segundos de al menos los cuatro checkpoints más largos durante el Intervalo de Medida.

3. El intervalo de checkpoint debe ser menor o igual que el Intervalo de Medida. Si el Intervalo de Checkpoint es menor que el Intervalo de Medida, el Intervalo de Medida debe ser un número entero del Intervalo de Checkpoint.

5.6 Informes Requeridos.

5.6.1 Se debe realizar un gráfico, de forma independiente para cada una de los cinco tipos de transacción (ej, New-Order, Payment, Order-Status, Delivery y Stock-Level), de la distribución frecuencial de los tiempos de respuesta de todas las transacciones, comenzadas y completadas durante el intervalo de medida. El eje x representa el TR de transacción y debe estar escalado desde 0 hasta cuatro veces el RT del 90 % por ciento de las transacciones para esa transacción. El eje y representa la frecuencia de las transacciones con un TR dado. Se deben incluir al menos 20 intervalos diferentes, de igual longitud. Se debe incluir también, el máximo, la media y el tiempo de respuesta del 90 % de las transacciones. Un ejemplo del gráfico se muestra en la figura A.5.

5.6.2 Se debe realizar un gráfico de tiempos de respuesta frente al rendimiento para la transacción New-Order, ejecutada con la mezcla requerida de la cláusula 5.2.3. El eje x representa el rendimiento de New-Orders. El eje y representa el Tiempo de Respuesta correspondiente al 90 % de las transacciones. Se debe dibujar un gráfico a aproximadamente el 50 %, 80 % y 100 % de la tasa de rendimiento declarada (los puntos adicionales son opcionales). Se deben medir los puntos del 50 % y 80 % la misma configuración que en la ejecución al 100 %, para un intervalo mínimo de 20 minutos, variando o bien el

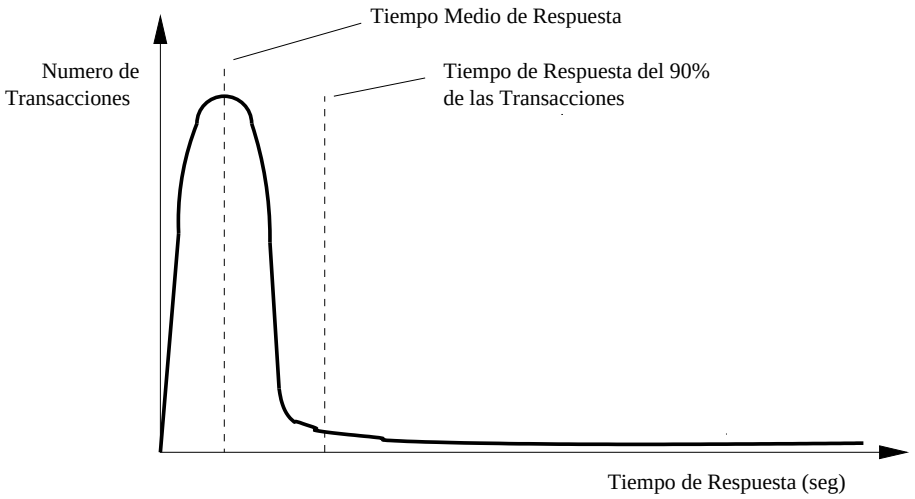


Figura A.5: Distribución del Tiempo de Respuesta de Transacción

Tiempo de Pensar de una o más tipos de transacción o bien el número de terminales activos. Se permite la interpolación del gráfico entre esos puntos. Se permite la variación de la mezcla de transacciones para los puntos del 50 % y el 80 %. Un ejemplo del gráfico se muestra en la figura A.6.

5.6.3 Se debe realizar un gráfico que muestre la distribución frecuencial de los Tiempos de Pensar para las transacciones New-Order, comenzadas y completadas durante el intervalo de medida. El eje x representa el Tiempo de Pensar y debe escalarse desde 0 hasta cuatro veces la media real del Tiempo de Pensar de esa transacción. El eje y representa la frecuencia de las transacciones con un Tiempo de Pensar dado. Se deben declarar al menos 20 intervalos diferentes de igual longitud. El tiempo medio del Tiempo de Pensar debe declararse también. Un ejemplo del gráfico se muestra en la figura A.7.

5.6.4 Se debe realizar un gráfico del rendimiento de las transacciones New-Order frente al tiempo transcurrido para tanto el periodo de rampa como para el intervalo de medida. El eje x representa el tiempo transcurrido desde el comienzo de la ejecución. El eje y representa el rendimiento en tpmC. Se deben usar al menos 240 intervalos diferentes con un tamaño máximo de intervalo de 30 segundos. Se debe señalar y mostrar también, en el gráfico, el comienzo y el fin del intervalo de medida. Se debe indicar en el gráfico el tiempo de comienzo para cada checkpoint. Un ejemplo del gráfico se muestra en la figura A.8.

5.7 Métricas Primarias.

5.7.1 Para estar de acuerdo con el estándar TPC-C y las Políticas de Uso Justo del TPC, todas las referencias públicas a los resultados TPC-C de una configuración deben incluir los siguientes componentes que se conocerán como métricas primarias.

- La tasa rendimiento máximo qualificado para el TPC-C (MQTh) expresada en tpmC. Se conoce como métrica de rendimiento. (Ver Cláusula 5.4)

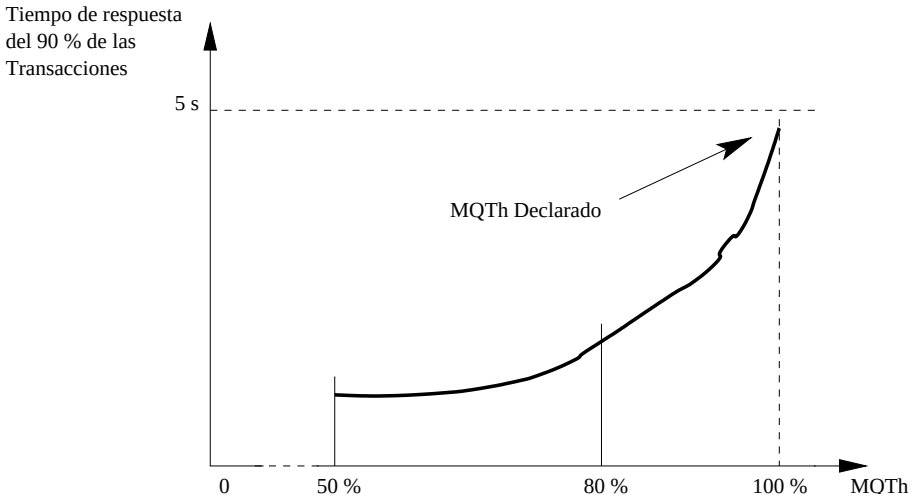


Figura A.6: Variación del rendimiento en función de la carga

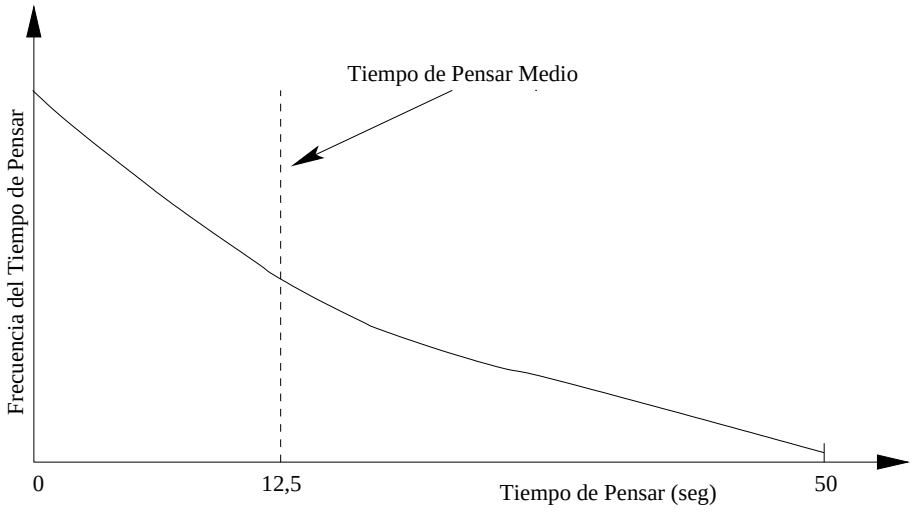


Figura A.7: Distribución del Tiempo de Pensar de transacción New-Order

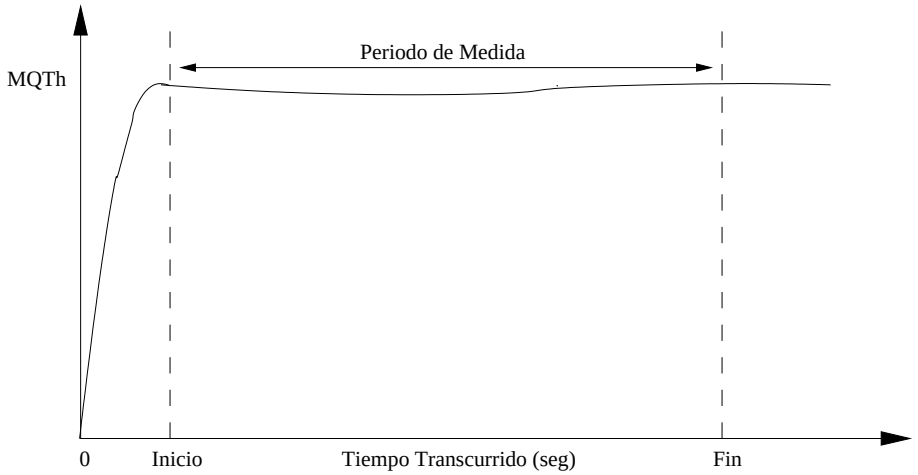


Figura A.8: Evolución del Rendimiento.

- El presupuesto TPC-C total a 3 años dividido por la MQTh y expresado como precio/tpmC. Se conoce también como métrica de Precio/Rendimiento. (Ver Cláusula 7.4)
- La fecha en la que todos los productos necesarios para alcanzar el rendimiento declarado están disponibles. Se conoce como fecha de disponibilidad. (Ver Cláusula 8.1.8.3)

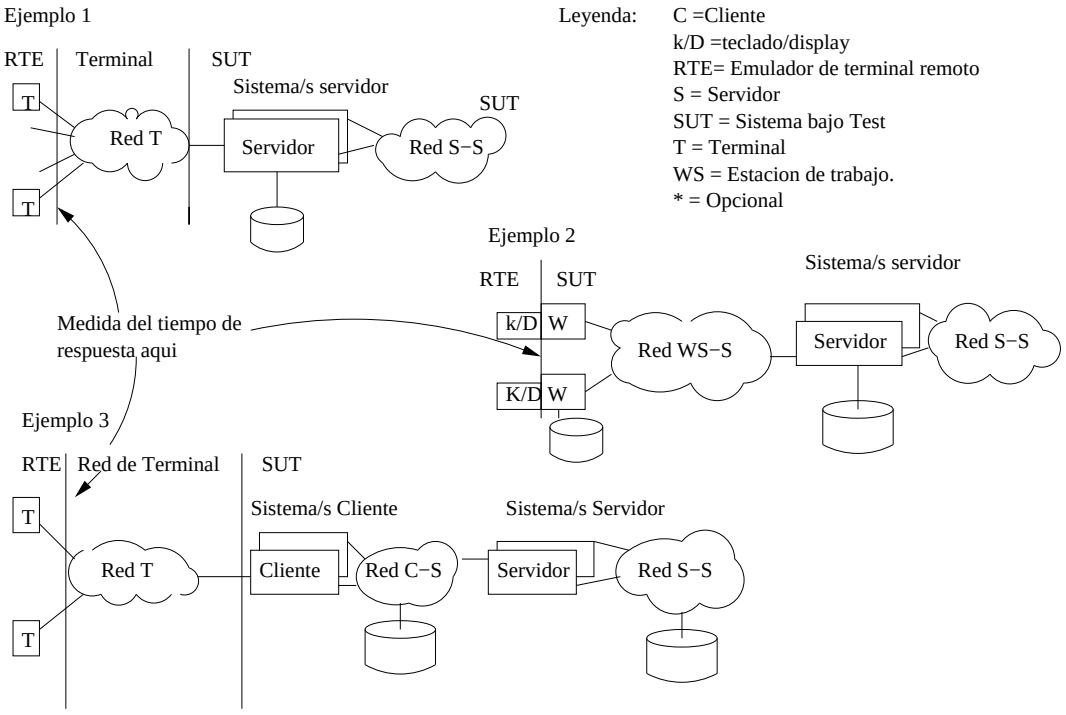


Figura A.9: Sistemas objetivo

Cláusula 6: Definición del SUT, el 'Driver' y las Comunicaciones

6.1 Modelos del Sistema Objetivo.

En la figura A.9 se muestran de forma gráfica algunos ejemplos de sistemas que representan el objetivo de este benchmark. A modo de explicación, las figuras distinguen también la frontera ETR/SUT (ver cláusulas 6.3 y 6.4) donde se mide el tiempo de respuesta.

6.2 Configuración del test.

La configuración del test se compone de los siguientes elementos:

- Sistema a prueba (SUT en inglés).
- Sistema(s) Driver.
- Interface de comunicaciones Driver/SUT.

Si una de las redes es una WAN, las configuraciones testeadas no necesitan incluir el trayecto de las líneas de comunicaciones.

6.3 Definición de Sistema bajo Test (SUT).

6.3.1 El SUT se compone de:

- Una o más unidades de proceso (ej., host, front-ends, estaciones de trabajo, etc.) en las que se ejecutarán la mezcla de transacciones descrita en la cláusula 5.2.3, y cuyo rendimiento global (Rendimiento Qualificado Máximo Total) estará reflejado en la medida de rendimiento tpmC.
- Se considera parte del SUT a cualquier sistema front-end. Los procesadores front-end de comunicación de datos, los controladores cluster, los clientes de base de datos (como es el modelo cliente/servidor), las y estaciones de trabajo son ejemplos de sistemas front-end.
- Los sistemas host, incluyendo el hardware y el software, que mantienen la base de datos empleada en el benchmark.
- Los componentes de hardware y software de todas las redes necesarias para conectar y mantener los componentes del SUT.
- El medio de almacenamiento suficiente para satisfacer tanto los requisitos de escalado de la cláusula 4.2 como las propiedades ACID de la cláusula 3.

6.3.2 Se puede usar un resultado de benchmark puntual para múltiples SUTs si se asegura el cumplimiento de las siguientes condiciones:

- Cada SUT debe tener la misma arquitectura hardware/software y la misma configuración.
- La única excepción permitida, sobre lo anterior, es diferir en los elementos que no están involucrados en el proceso lógico del SUT (ej., numero de ranuras para periféricos, alimentación, embalado, ventiladores, etc.)
- Todos los SUT deben soportar la configuración propuesta.

6.4 Definición de Driver.

6.4.1 Se debe usar un sistema(s) Driver, que proporcione(n) las funciones de un Emulador de Terminal Remoto (ETR), a fin de emular la población de terminales y con sus usuarios emulados durante la ejecución del benchmark.

6.4.2 El ETR realiza las siguientes funciones:

- Emula la introducción de datos en la pantalla de entrada/salida de un terminal emulado, generando y mandando mensajes de transacción al SUT.
- Emula la muestra de mensajes de salida en una pantalla de entrada/salida, recibiendo mensajes de respuesta del SUT.
- Registra los tiempos de respuesta.
- Realiza la conversión o la desmultiplexación según protocolo de comunicaciones usado en la interfaz de comunicaciones entre el driver y el SUT.

- Realiza los cálculos estadísticos (ej., el cálculo del tiempo de respuesta del 90 %, cálculo del rendimiento, etc.).

6.4.3 Normalmente se espera que el sistema Driver realice funciones del RTE exclusivamente. El trabajo que se realice en el Driver además del especificado en la cláusula 6.4.2 debe ser justificado exaustivamente como se especifica en la cláusula 6.6.3.

6.4.4 El motivo de lo anterior es que el Sistema Driver debe reflejar la configuración de terminales propuesta y no debe añadir funcionalidad o rendimiento por encima de los componentes de red propuestos para el SUT. Se debe demostrar que los resultados de rendimiento no se aumentan con el uso del sistema Driver.

6.4.5 El Software o el hardware que reside en el Driver que no sea el ETR se considera parte del SUT. Por ejemplo, en un modelo cliente/servidor, el software del cliente puede ser ejecutado o simulado en el sistema Driver (ver cláusula 6.6.3).

6.5 Definiciones de Interfaces de Comunicaciones.

6.5.1 Conexiones de Canales de E/S.

6.5.1.1 Todos los protocolos utilizados deben estar disponibles en el mercado.

Comentario: La intención de esta definición es excluir conexiones de canales de E/S no estándar. Los siguientes casos son ejemplos de conexiones de canales aceptadas:

- Configuraciones o arquitecturas donde los terminales o los controladores de terminales están conectados normalmente o rutinariamente a una canal de E/S de un procesador.
- Donde el(los) procesador(ores) en el SUT está(n) conectados a la red de comunicaciones local por medio de un procesador front-end conectado a través de un canal. El procesador front-end debe incluirse en el precio del SUT.

6.5.2 Interfaz de comunicaciones Driver/SUT.

6.5.2.1 La interfaz de comunicaciones entre el(los) sistema(s) Driver y el SUT debe ser el mismo mecanismo a través del cual el sistema estaría conectado con el terminal (ver cláusula 2.1.8) en la configuración propuesta.

6.6 Requisitos Añadidos del SUT y del Sistema Driver.

6.6.1 Restricciones del Sistema Driver.

Las copias de alguna parte de la base de datos o sistema de ficheros a prueba o sus estructuras de datos, índices, etc. no deben estar presentes en el Sistema Driver durante el test.

Comentario: No se permite la sincronización entre el ETR y el SUT.

6.6.2 Contextos Individuales para los Terminales Emulados.

El SUT debe contener un contexto para cada terminal emulado, y debe mantener ese contexto durante la duración del test. Ese contexto debe ser idéntico al que tendría con el terminal real. Un terminal que manda una transacción no debe mandar otra hasta que se complete la anterior, con la excepción de la ejecución aplazada de la transacción Delivery.

Comentario: El contexto al que se refiere esta cláusula debe consistir en información tal como la identificación del terminal, identificación de la red, y otra información necesaria para que un terminal real sea reconocido (o configurado) por el SUT. La intención es permitir transacciones pseudo- conversacionales. El objetivo de esta cláusula, es simplemente, evitar que un sponsor multiplexe mensajes de un conjunto muy amplio de terminales emulados en unas pocas líneas y asegure que el sistema soporta ese número de usuarios sin considerar si el sistema soporta realmente ese número de terminales reales. Se permite que un terminal pierda su conexión con el SUT durante el intervalo de Medida siempre y cuando no se pierda su contexto y se reconecte en un máximo de 90 segundos usando el mismo contexto. La perdida y la reconexión de un usuario deben ser anotadas y exponer su número total.

6.6.3 Sistema Driver que hace más que un ETR.

Se debe justificar el uso de Sistema Driver para emular funciones adicionales diferentes que las descritas en la Cláusula 6.4, como sigue:

6.6.3.1 Se debe demostrar que la arquitectura de la solución propuesta hace inviable económicamente el configurar el benchmark para evitar que el driver realice el trabajo en cuestión. (p.e, en una base de datos cliente/servidor, donde el software del cliente se ejecutaría en un número amplio de estaciones de trabajo).

6.6.3.2 No se debe violar la regla 6.6.1.

6.6.3.3 Se debe demostrar que los ejecutables localizados en el Sistema Driver son equivalentes en cuanto al funcionamiento que los que existen en el sistema propuesto.

6.6.3.4 Se debe demostrar que los resultados de rendimiento no se mejoran al realizar el trabajo en cuestión en el Sistema Driver. Se debe ejecutar un test para demostrar que la funcionalidad, el rendimiento y la conectividad de la solución emulada es la misma que de la que se da el precio. Los datos del test deben incluirse en el Informe Completo de Especificaciones. Por ejemplo, si el Sistema Driver emula las funciones de un concentrador de terminales, debe haber datos de test que demuestren que, un concentrador real con el número de terminales especificado conectados al él, ofrecería el mismo (o mejor) tiempo de respuesta que el medido con el Sistema Driver. El concentrador de terminales debe estar configurado como si formara parte del sistema propuesto y cargado con el número máximo de líneas usadas en la configuración propuesta. El test de demostración debe ejecutarse como parte de la configuración del SUT cuando esté trabajando a plena carga en una base de datos con la escalada apropiada. El siguiente diagrama de la figura A.10 se ilustra la configuración de un posible test de demostración.

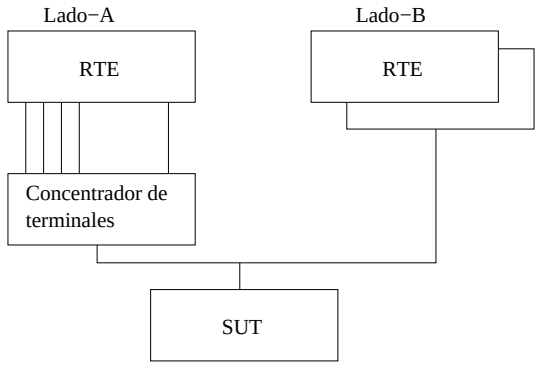


Figura A.10: Ejemplo de configuración

En el ejemplo de la figura A.10, la diferencia del tiempo de respuesta entre la parte A y la parte B debería ser menor o igual que cualquier ajuste del tiempo de respuesta indicado en el Informe Completo de Especificaciones.

Si se usa el retardo del tiempo de respuesta de un test de demostración en múltiples pruebas de benchmark, la demostración debe realizarse en un SUT que genere la tasa más alta de tpmC en el emulador de terminal.

6.6.3.5 Deben mantenerse los contextos individuales del ETR hacia el SUT.

6.6.3.6 Deben incluirse diagramas completos de tanto la configuración del benchmark como la configuración del sistema propuesto. Debe incluirse una lista detallada de todas las funciones software/hardware que se realizan en el Sistema Driver, así como la interfaz con el SUT.

6.6.3.7 Cuando se emulen dispositivos de usuario final usando un navegador web, se debe incluir 0,1 segundos al tiempo de respuesta en la emulación para compensar el retardo que se encuentra en la configuración extremo a extremo debido al retardo del navegador.

6.6.4 Descripción de la Configuración de Red y las Partes Emuladas.

El patrocinador del test deberá describir completamente las configuraciones de red de tanto los servicios puestos a prueba como los servicios reales propuestos que se van a entregar. Se debe dar una explicación exhaustiva de exactamente qué partes de la configuración propuesta se van a reemplazar por el Sistema Driver.

6.6.5 Límites de Concentración.

El nivel de concentración de mensajes entre el sistema Driver y el SUT en la configuración del benchmark no debe exceder de aquel que se daría en la configuración propuesta. En particular, el número de paquetes de comunicación que se pueden concentrar, no debe exceder del número de terminales que estarían conectados directamente a ese concentrador en la configuración propuesta.

6.6.6 Limites en La Intervención del Operador.

Se permiten aquellos sistemas que precisan de una intervención del operador durante las operaciones normales para mantener el rendimiento declarado para un periodo de ocho horas, siempre y cuando cumplan las siguientes condiciones:

- Se debe justificar la necesidad de la intervención del operador en el Informe Completo de Especificaciones. El informe debe describir las funciones realizadas por el operador y la frecuencia de esta actividad.
- Se debe describir el mecanismo por el cual el sistema avisa que se necesita la intervención del operador. Este mecanismo debe implementarse por medio de hardware y software disponible en el mercado y debe incluirse en el precio del SUT.
- Cualquier evento (o combinación de eventos) que precisen la intervención del operador debe(n) ofrecer al operador la posibilidad de responder dentro de al menos 30 minutos antes de que el evento pueda afectar al rendimiento del sistema.

Comentario: el objetivo de esta cláusula es restringir el número de intervenciones del operador a un nivel razonable. Debido a la naturaleza critica de la aplicación OLTP modelada, no se permiten los sistemas que dependan de una intervención rápida del operador para mantener el rendimiento.

Apéndice B

Herramientas SQL

En este apéndice se hará una explicación del lenguaje que se ha utilizado para implementar las operaciones que el benchmark realiza contra la base de datos así como del modelo en que ésta se basa.

Dado que el lenguaje SQL se basa en el modelo relacional, primeramente se hará una descripción de los principios en el que éste se basa, para después hablar de SQL.

B.1. El modelo relacional de bases de datos

B.1.1. Principios del modelo relacional

El modelo relacional nos ofrece una manera única de representar los datos: una tabla bidimensional denominada *relación*. A continuación se describen brevemente algunos términos relacionados con este modelo:

- Una *relación* se corresponde con lo que podríamos llamar una tabla.
- Una *tupla* se corresponde con una fila de esa tabla. El número de filas se conoce como *cardinalidad*.
- Un *atributo* se corresponde con una columna de esa tabla. El número de columnas se conoce como *grado*.
- La *clave primaria* (o *llave primaria*) es un identificador único para la tabla; es decir, es una columna o combinación de columnas con la siguiente propiedad: nunca existen dos filas de la tabla con el mismo valor en esa columna o combinación de columnas o lo que es lo mismo con el mismo valor de llave primaria.
- Un *dominio* es el conjunto o conjuntos de valores de los cuales los atributos toman sus valores.

A continuación se muestra un ejemplo aclaratorio que servirá para entender mejor el modelo relacional. Se representará una relación a la que llamará 'Almacén' y que tiene el siguiente trazado:

NOMBRE	ARTÍCULOS	APERTURA
Nombre1	1500	12-6-99
Nombre2	900	27-7-01
Nombre3	2100	08-3-97
Nombre4	1700	25-6-00

La relación 'Almacén' se compone de tres atributos: NOMBRE (el nombre del almacén), ARTÍCULOS (el número de artículos en ese almacén) y APERTURA (la fecha de apertura de ese almacén). La cardinalidad de esta tabla es cuatro y su grado es tres.

B.1.2. Esquemas

El nombre de una relación y el conjunto de sus atributos reciben el nombre de *esquema* de la relación. El esquema de la relación se denota con el nombre de ésta seguido de una lista de sus atributos entre paréntesis. Así el esquema de la relación 'Almacén' del ejemplo anterior es:

Almacén (Nombre, Artículos, Apertura)

B.1.3. Dominios

Antes de describir más en profundidad el concepto de dominio, se establecerá cuál es la menor unidad semántica de información. Los valores *escalares* son la menor unidad semántica de información, en el sentido de que son atómicos; no poseen estructura interna (no se pueden descomponer sin que con ello pierdan información) desde el punto de vista del modelo.

Tras esto se define *dominio* como un conjunto de valores escalares todos del mismo tipo. Son los grupos de valores de los cuales se extraen los valores que aparecen en los atributos. Así por ejemplo en el ejemplo anterior el dominio del atributo NOMBRE lo compondrían todos los nombres posibles de almacén que podrían aparecer en esa columna.

B.1.4. Relaciones o tablas

A continuación se definirá el término *relación* de una forma más específica. Una relación (por ejemplo R) sobre un conjunto de dominios D1, D2,..Dn (no necesariamente todos distintos), se compone de una cabecera y un cuerpo.

- La cabecera está formada por un conjunto fijo de atributos, o, en términos más

precisos, de pares atributo/dominio:

(A1:D1),(A2:D2),...(An:Dn)

tales que cada atributo Aj correspondo a uno y sólo uno de los dominios subyacentes

Dj(j=1,2,...,n)

- El cuerpo está formado por un conjunto de tuplas, el cual varía con el tiempo. Cada tupla a su vez está formada por un conjunto de pares atributos/valor:

  (A1:vi1),(A2:vi2),...,(An:vin)

  con i=1,2,..,m, donde m es el número de tuplas del conjunto. En cada una de estas tuplas hay uno de estos pares atributo/valor (Aj:vij) para cada atributo 'Aj' de la cabecera. Para cada par atributo/valor (Aj:vij), 'vij' es un valor del dominio único Dj asociado al atributo Aj.

Los valores de m y n se llaman cardinalidad y grado, respectivamente, de la relación R. La cardinalidad varía con el tiempo pero el grado no.

B.1.5. Llaves primarias

Decimos que un conjunto de atributos A1,A2,..,An es una llave primaria de una relación R si:

1. Los atributos determinan funcionalmente al resto de los atributos de la relación. En otras palabras, es imposible que dos tuplas distintas de R coincidan en todos los atributos A1, A2, ..An.
2. Ningún subconjunto propio de A1, A2,..,An determina funcionalmente al resto de los atributos de la tabla R, es decir, una llave debe ser mínima. Por lo tanto los atributos de una llave identifican unívocamente al resto de atributos de cada fila.

Ejemplo: En el ejemplo anterior podríamos definir como llave primaria de la relación al atributo NOMBRE. Esto haría que este atributo definiera unívocamente al a cada fila de la relación.

B.2. El lenguaje SQL (Lenguaje de Consulta Estructurado)

Los sistemas de bases de datos relacionales más comunes consultan y modifican la base de datos por medio del lenguaje SQL (*Structured Query Language* o Lenguaje de Consulta Estructurado). Buena parte de lenguaje se basa en el álgebra relacional, aunque ofrece muchas características importantes que van más allá de lo ésta ofrece:

por ejemplo, la agregación (sumas y conteos, entre otras cosas) y las actualizaciones de la base de datos.

Este lenguaje consta de muchos dialectos, existiendo dos principales: el SQL ANSI (*American National Standards Institute*) y un estándar actualizado que se adoptó en 1992, llamado SQL-92 o SQL2.

En los siguientes apartados se explicará el uso del SQL como un lenguaje autónomo de consultas, es decir: como se usaría en un terminal donde se ejecutan consultas y modificaciones sobre la base de datos. Al final se explicará cómo puede usarse el SQL enmarcado dentro de otro lenguaje de programación, lo que se conoce como SQL embebido.

En este proyecto se ha utilizado el lenguaje SQL para interactuar con la base de datos. Aunque el SQL ofrece muchas posibilidades, los siguientes apartados básicamente se centrarán en las características que se han utilizado para la implementación del benchmark.

B.2.1. Consultas en SQL

Una de las consultas más simples que se pueden realizar en el lenguaje SQL es solicitar las tuplas de una relación que cumplan con una condición. Esta consulta se realiza a través de tres palabras clave: `SELECT`, `FROM` y `WHERE`.

Por ejemplo, en el sistema de base de datos utilizado en el benchmark podemos solicitar las tuplas de la relación `item` cuyo atributo `i_id` es mayor que 5000 e `i_price` es menor que 150 a través de la siguiente consulta:

```
SELECT *
FROM item
WHERE i_id > 5000 AND i_price < 150;
```

donde:

- En la clave `FROM` se señalan las relaciones a las que se va a hacer referencia en la consulta. En este caso la relación `item`.
- En la clave `SELECT` se indican los atributos de las relaciones que se indican en la clave `FROM`. Estos atributos son los que aparecerán en la consulta. El signo `'*'` de este ejemplo indica se desea que se muestren todos los atributos de la tabla `item`.
- En la clave `WHERE` se indica la condición o condiciones que deben cumplir las tuplas cuyos atributos queremos que se muestren. En este caso las tuplas deben tener un valor en el atributo `i_id` mayor que 5000 y un valor en el atributo `i_price` menor que 150.

La clave `WHERE` se puede omitir. En ese caso se mostrarían todas las tuplas de la relación indicada.

Proyección en SQL

Si no se desea que se muestren todos los atributos de la relación especificada, se pueden eliminar los atributos no deseados a través de una *proyección*. Las proyecciones en SQL se realizan indicando en la cláusula SELECT los atributos que desea que se muestren. Por ejemplo, en el sistema de base de datos del benchmark se podrá realizar la siguiente proyección:

```
SELECT c_id,  c_first, c_middle, c_last
FROM customer
WHERE c_d_id > 16;
```

Esta consulta produciría como resultado las tuplas con los nombres y apellidos de los clientes que pertenezcan a los distritos cuyo número de identificación es mayor que 16.

C_ID	C_FIST	C_MIDDLE	C_LAST
1601	JOHN	AN	BAROUGTHABLE
1602	MARK	BZ	PRESBARCALLY
1603	RICHARD	JK	ANTIBAREING
.	.	.	.
.	.	.	.
.	.	.	.

Si queremos que el resultado tenga los nombres de la columna diferentes a los nombres de los atributos, se puede escribir en la clave `SELECT`, detrás del nombre de atributo, la palabra clave `AS` seguido del alias que se desea poner al atributo. Las siguientes líneas:

```
SELECT c_id AS Codigo, c_first AS Nombre , c_middle AS Ape1, c_last AS Ape2
FROM customer
WHERE c_d_id >16;
```

producirían como resultado:

CODIGO	NOMBRE	APE1	APE2
1601	JOHN	AN	BAROUGTHABLE
1602	MARK	BZ	PRESBARCALLY
1603	RICHARD	JK	ANTIBAREING
.	.	.	.
.	.	.	.
.	.	.	.

Selección en SQL

La selección en SQL se realiza gracias a la clave WHERE. En las expresiones de selección se pueden utilizar los operadores =, <>, <, >, <= y >=. El significado de los operadores, exceptuando <>(no igual que), es evidente.

Se pueden utilizar en las expresiones de selección formulas aritméticas como, la suma (+), la resta (-) y la multiplicación (*).

Un ejemplo de selección es:

```
SELECT c_id AS CODIGO,  c_first AS NOMBRE
FROM  CUSTOMER
WHERE c_d_id  > 16 AND c_last = 'PRESBARCALLY';
```

que produciría como resultado:

CODIGO	NOMBRE
1602	MARK

El resultado de una operación de comparación es un valor lógico: VERDADERO o FALSO, por lo tanto se pueden combinar las expresiones utilizando los operadores lógicos AND, OR y NOT. AND tiene preferencia ante OR y NOT ante ambos.

Comparación de cadenas

En SQL dos cadenas son iguales si contienen la misma secuencia de caracteres. Sin embargo, SQL permite también la comparación entre cadenas de tamaño variable que se rellenan con espacios y cadenas de longitud fija. Por ejemplo, la expresión Pepe = Pepe, daría como resultado VERDADERO.

Al utilizar los operadores >o <para comparar cadenas, tenemos que tener en cuenta que SQL compara las cadenas utilizando el orden alfabético. Por lo tanto, 'hola' >'adios' y 'soli' <'soliloquio'.

Comparación de fechas y horas

Para comparar fechas y horas se pueden utilizar, al igual que en las cadenas, los símbolos '<' y '>', de manera que 'a >b' si 'a', es una fecha más temprana que 'b'.

Ordenación de la Salida

En ocasiones se necesita que las tuplas se ordenen de según el valor de los atributos. Esto se puede obtener añadiendo a la estructura de selección descrita anteriormente la sentencia `ORDER BY` seguida por los atributos que se necesita para ordenar las tuplas. El procedimiento es ordenar las tuplas utilizando el atributo situado en primer lugar. Si se dan coincidencias, se resuelven gracias al segundo atributo y si de nuevo se dan coincidencias se resuelven sucesivamente con el tercero, cuarto, etc.

El ejemplo:

```
SELECT c_id, c_first, c_middle, c_last
FROM customer
WHERE i_id >1600 AND c_id <1603
ORDER BY c_last;
```

produce:

CODIGO	NOMBRE	APE1	APE2
1603	RICHARD	JK	ANTIBAREING
1601	JOHN	AN	BAROUGTHABLE
1602	MARK	BZ	PRESBARCALLY

Productos en SQL

SQL permite realizar operaciones algebraicas sobre varias relaciones, tales como productos de conjuntos. En SQL se pueden acoplar varias relaciones, añadiendo el nombre de las relaciones utilizadas a la clave `FROM` y utilizando cualquier atributo de las tablas en las claves `SELECT` y `FROM`. Por ejemplo, si necesitamos saber el nombre del artículo de la fila 5 de la orden 10.000, realizaremos la siguiente consulta:

```
SELECT i_name
FROM order-line, item
WHERE ol_o_id = 10000 AND ol_number = 5 AND ol_i_id = i_id;
```

La consulta devuelve el nombre del artículo cuyo número de identificación es igual al `ol_i_id` de la tupla número 5 de la tabla `order_line` que corresponde a la orden 10.000.

Duplicados

En ocasiones las operaciones realizadas en las relaciones provocan como resultado listados en los que algunos de los elementos aparecen repetidos. En este apartado se verá como se puede evitar este fenómeno.

SQL permite eliminar los duplicados en operaciones de selección y de proyección de productos cartesianos añadiendo en la clave `SELECT` la palabra clave `DISTICT`. Utilizando el sistema de base de datos del benchmark, si se desea obtener los nombres de los artículos que se han vendido realizaremos se puede utilizar la consulta:

```
SELECT DISTINCT i_name
FROM order-line, item
```

```
WHERE ol_i_id = i_id;
```

Si se omitiera la palabra `DISTINCT` se obtendría un listado en el que probablemente aparecieran repetidos los nombres de varios artículos ya que pueden haber varias ordenes que contengan dicho artículo.

Agregación

SQL permite añadir una columna que contenga valores calculados a partir de los valores de alguna columna. También es posible añadir a cada tupla de la relación un campo con el resultado de una operación en la que intervengan los atributos de la tupla. Ejemplos de lo anterior son: calcular el valor medio de los valores de una columna, el valor máximo de los valores de una columna o la suma para cada tupla de dos de sus atributos.

Los operadores de Agregación son:

- `SUM`, suma de los valores de una columna.
- `AVG`, cálculo del valor medio de los valores de una columna.
- `MIN`, cálculo del valor mínimo de los valores de una columna.
- `MAX`, cálculo del valor mínimo de los valores de una columna.
- `COUNT`, número de los valores de una columna;

Los operadores anteriores se utilizan en una sentencia `SELECT`, `FROM`,`WHERE` detrás de la clave `SELECT`.

Ejemplo: Para contar el número de artículos distintos de la relación `item`, se usaría la siguiente sentencia:

```
SELECT DISTICT COUNT(i_name)
FROM item;
```

Nota: se antepone a `COUNT` la palabra `DISTINCT` ya que la relación `ITEM` podría contener dos artículos con el mismo nombre.

Agrupamiento

Es muy probable que no se quiera obtener una agregación que tenga en cuenta las tuplas de una relación en su totalidad, sino que se desee que esa agregación afecte a las

tuplas agrupadas según un determinado atributo. Por ejemplo, por medio de la siguiente consulta, se puede calcular la suma de los precios de todos los artículos suministrados por cada almacén:

```
SELECT SUM(ol_amount)
FROM order-line
GROUP BY ol_supply_w_id;
```

Restricción de grupos

Se puede modificar la consulta anterior para que se muestren únicamente la suma de los precios de los artículos vendidos en los grupos que tengan artículos con precios mayores que 500:

```
SELECT SUM(ol_amount)
FROM order-line, item
WHERE ol_i_id = i_id
GROUP BY ol_supply_w_id
HAVING MIN(i_price) >500;
```

La clave `HAVING` restringe los grupos cuyo valor mínimo en el atributo `i_price` sea mayor que 500.

B.3. Modificaciones de la base de datos

Las operaciones de modificación de la base de datos son la adicción, eliminación y actualización de alguna o algunas de las filas de las tablas que existen en la base de datos.

B.3.1. Inserción de filas

Los formatos de la consulta SQL para insertar filas en una tabla con nombre `tabla1` son los siguientes:

```
INSERT FROM tabla1(atributo1, atributo2, ..., atributoN)
```

o bien:

```
INSERT FROM tabla1
VALUES (atributo1, atributo2, ..., atributoN);
```

El primer formato inserta una fila cuyos atributos toman los valores que se indican entre paréntesis. Si algún atributo se omite, en el lugar correspondiente a ese atributo dentro de la base de datos se almacena un valor NULL.

El segundo formato inserta una fila cuyos atributos toman los valores que se indican entre paréntesis después de la palabra clave VALUES. Al igual que en el formato anterior si algún atributo se omite, en su lugar se almacena un valor NULL.

A pesar de lo que pueda parecer, no es necesario conocer los atributos a priori, sino que los atributos pueden proceder de una subconsulta, como es el caso del siguiente ejemplo:

```
INSERT INTO order-line (ol_i_id)
SELECT i_id
FROM item
WHERE i_name = articulo1;
```

En este ejemplo se inserta una fila en la tabla `order_line` con todos los atributos con el valor NULL excepto `i_i_id` que coincide con el número de identificador del artículo cuyo nombre es `articulo1`.

B.3.2. Eliminación de filas

El formato de la consulta SQL para eliminar filas en una tabla con nombre `tabla1` es el siguiente :

```
DELETE FROM tabla1
WHERE <Condición>;
```

Esta consulta elimina todas las filas de la tabla `tabla1` que cumplan con la condición especificada en <Condición>. Un ejemplo es eliminar todas las filas de la tabla `item` que correspondan a artículos con precio mayor que 500:

```
DELETE FROM item
WHERE i_price >500;
```

B.3.3. Actualización de filas

Una actualización es el proceso que permite la modificación de algún o algunos de los atributos de las tuplas pertenecientes a una relación.

El formato de una consulta de actualización en SQL es el siguiente:

```
UPDATE <tabla>
SET <nuevos valores>
WHERE <condición>;
```

donde:

- `<tabla>` es la relación que contiene las tuplas que se desea modificar.
- `<nuevos valores>` se asignan aquí los valores nuevos para cada atributo utilizando la sintaxis `<nombre_atributo>= <nuevo valor>`.
- `<condicion>` condición que deben cumplir las tuplas de `<TABLA>` para que puedan ser modificadas.

Ejemplo: Para insertar el valor `'GC'` en el atributo `c_credit` de todas las tuplas de la tabla `customer` correspondientes a los clientes con un `c_balance <-1000`, se puede usar la siguiente sentencia.

```
UPDATE customer
SET c_credit = 'GC'
WHERE c_balance <-1000
```

B.4. Definición del esquema relacional en SQL

A continuación se explicarán los procedimientos en SQL que permiten describir la estructura de la información de la base de datos, esto es, definir los atributos de cada tabla y su tipo de datos asociado.

Tipos de datos de los atributos.

1. Cadenas de caracteres. En SQL hay dos tipos de cadenas de caracteres:
   - *a*) Cadenas fijas: el número de caracteres que contienen es constante. En SQL se representa con la sintaxis `CHAR(n)`, donde 'n' es el número de caracteres de la cadena.
   - *b*) Cadenas variables: pueden contener cualquier secuencia variable de caracteres, con un tamaño máximo de n. La sintaxis para definir este tipo de datos en SQL es `VARCHAR(n)`.
2. Cadenas de bits. Al igual que las cadenas de caracteres, SQL tiene dos variantes:

   a) Cadenas fijas: Se definen de la forma `BIT(n)`, y representan una cadena fija de n bits.

   b) Cadenas variables: son una ristra de bits con un tamaño máximo de n, se declaran con la sintaxis `BIT VARYING (n)`.

3. Enteros. La declaración de este tipo se hace por medio de las palabras claves `INT` o `INTEGER`. Algunas implementaciones también permiten definir el tipo `SHORTINT` que representa a un entero de valor menor que `INT`.

4. Números en coma flotante. Hay varias variantes:

   a) `FLOAT` o `REAL`: representan números en coma flotante comunes.

   b) texttt`DOUBLE PRECISION`: representan números en coma flotante con precisión extendida.

   c) texttt`DECIMAL (n, d)`: declara un tipo de datos que puede contener números de n dígitos enteros y d dígitos decimales.

5. Fechas y horas. Se pueden definir fechas utilizando la palabra clave `DATE`. Para las se utilizará la palabra clave `TIME`.

B.4.1. Declaración de tablas

Una tabla en SQL se define con la siguiente sintaxis:

```
CREATE TABLE <nombre\_tabla> (
          <atributo1> <tipo de datos>;
          <atributo2> <tipo de datos>;
          <atributo3> <tipo de datos>;
              .              .
              .              .
              .              .
          <atributoN> <tipo de datos>
);
```

Ejemplo: definición de la tabla `item` en la base de datos del benchmark:

```
CREATE TABLE item (
i\_id     DECIMAL(6,0);
i\_im\_id DECIMAL(6,0);
i\_name   VARCHAR(24);
i\_price  DECIMAL(5,0);
i\_data   VARCHAR(50);
);
```

B.4.2. Eliminación de tablas

Si por cualquier circunstancia se hace necesaria la eliminación de una tabla de la base de datos, SQL tiene una sentencia que permite realizar esta operación:

```
DROP <nombre_tabla>;
```

B.4.3. Modificación de las tablas.

SQL permite añadir o eliminar atributos por medio de las siguientes consultas:

- Para añadir atributos:

  `ALTER TABLE` <nombre_tabla>`ADD` <nombre_atributo><tipo_datos>;

- Para eliminar atributos:

  ALTER TABLE <nombre_tabla>DROP <nombre_atributo><tipo_datos>;

Ejemplo: se puede añadir un campo en la tabla `item` que indique la fecha de adquisición del artículo mediante la consulta:

```
ALTER TABLE item ADD i_fecha_adq DATE;
```

B.4.4. Creación de índices

Un índice es un mecanismo interno a la base de datos que permite acelerar la búsqueda de una tupla cuando se seleccione por medio del atributo al que hace referencia el índice. Por ejemplo, si definimos un índice en la tabla `ITEM` para el atributo `i_id`, el proceso de búsqueda de un artículo, por su número de identificación, será más rápido que antes de definir el índice. SQL permite definir índices utilizando el siguiente formato para la consulta:

```
CREATE INDEX <nombre_indice>ON <nombre_tabla>(<atributo1>, <atributo2>...,
<atributoN>);
```

Como puede verse, un índice puede hacer referencia a una combinación de atributos. Aunque esto no es usual, resulta bastante útil en los índices que referencian a llaves primarias múltiples.

También es posible eliminar índices mediante la consulta:

```
DROP INDEX <nombre_indice>;
```

Ya que los índices facilitan enormemente la labor en las búsquedas, se podría considerar la creación de un índice para cada atributo de cada tabla, sin embargo hay que tener

en cuenta que un índice es una estructura de datos dificulta y ralentiza las inserciones, eliminaciones y actualizaciones de las tuplas. Cuando se diseñe una base de datos hay que llegar a un compromiso entre la velocidad las búsquedas búsqueda y el coste de la modificación de las tablas.

B.5. Restricciones en SQL

B.5.1. Introducción

Uno de los problemas más serios de quienes escriben aplicaciones que actualizan la base de datos es el de que la nueva información puede ser errónea en varios aspectos. La forma más simple de cerciorarse de que las modificaciones de la base de datos no permitan tuplas inapropiadas en las relaciones consiste en escribir programas de aplicación de tal manera que todo comando de inserción, eliminación, y actualización incluya las verificaciones necesarias que garantizan su corrección. Por desgracia, los requisitos de corrección a menudo resultan complejos y siempre son repetitivos; los programas de aplicación han de repetir las misma pruebas tras cada modificación.

EL SQL ofrece varios métodos para expresar las restricciones de integridad como parte del esquema de la base de datos. En la presente sección se estudiarán las principales como son las restricciones de llaves, y la integridad referencial.

B.5.2. Llaves en SQL

Una de las restricciones para una base de datos consiste en que un atributo o conjunto de atributos constituye una llave de una relación. En otras palabras se prohibe que dos tuplas de la relación concuerden en el atributo o atributos que han sido declarados como llave primaria. Una llave primaria se declara dentro del comando `CREATE TABLE` de SQL. Hay dos formas similares de declarar llaves: por medio de las palabras clave `PRIMARY KEY` o mediante la palabra clave `UNIQUE`. Con todo una tabla puede tener sólo una llave primaria y varias declaraciones únicas.

Mediante la declaración `CREATE TABLE` se tienen dos formas de declarar la llave primaria:

- Se puede definir el atributo como llave primaria, añadiendo en su declaración las palabras `PRIMARY KEY`. Ejemplo:

```
CREATE TABLE Almacén (
        nombre CHAR(10) PRIMARY KEY
        articulos       INT
        fecha           DATE
);
```

- Se puede añadir a la declaración de la tabla una declaración más donde que diga que un atributo particular o un conjunto de atributos constituyen una llave primaria. Ejemplo:

```
CREATE TABLE Almacén (
        nombre    CHAR(10)
        articulos INT
        fecha     DATE
        PRIMARY   KEY (nombre)
);
```

En el caso de que se desee declarar más de un atributo como llave primaria tendremos que usar el segundo método. Si por ejemplo se quiere declarar como llave primaria a los atributos nombre y fecha se sustituirá la última línea por:

```
PRIMARY KEY (nombre, fecha);
```

Otra manera de crear una llave es mediante la palabra clave `UNIQUE` que se puede usar de las mismas maneras que `PRIMARY KEY`. `UNIQUE` tiene la misma importancia que la declaración de llaves primarias con la excepción de una relación puede haber varias declaraciones `UNIQUE` pero sólo una llave primaria. Podemos usar una llave `UNIQUE` para un único atributo cuando queremos que los valores de ese atributo sólo aparezcan una vez en toda la relación.

B.5.3. Integridad referencial y llaves exteriores

La integridad referencial establece que si una determinada relación 'R' contiene un atributo 'Rj' con un valor determinado 'c' este valor no debe ser cualquiera si no que debe aparecer en otro atributo de otra relación determinada, por ejemplo que el valor c aparezca en el atributo Si de la relación 'S'. Decimos entonces que el atributo 'Rj' es una llave exterior.

Declaración de restricciones y llaves exteriores.

En SQL se puede declarar a un atributo u atributos como llave exterior (o llave foránea), al hacer referencia a un atributo o atributos de una segunda relación (que puede ser la misma). Esto tiene como consecuencia:

- El atributo/s referenciados en la segunda relación han de ser llaves primarias de ésta.
- Cualquier valor que aparezca en un atributo de la llave exterior en la primera relación también habrá de aparecer en al atributo correspondiente de la segunda relación. Es decir, se da una restricción de integridad referencial que conecta ambos atributos o conjuntos de atributos.

Ejemplo: Supóngase la relación 'Almacén' descrita anteriormente, y una nueva relación a la que llamaremos 'Empleados':

```
CREATE TABLE Empleados (
        nombre2 CHAR(10)  /*nombre del almacén*/
        empleados int     /*número de empleados del almacén*/
);
```

Se puede pensar que todos los nombres de almacén de la relación `Empleados` han de aparecer en la relación `Almacén`, con lo cual podemos declarar al atributo `nombre2` de la tabla `Empleados` como una llave exterior que referencia a `nombre` de `Almacén`.

Existen dos formas de declarar llaves exteriores en la declaración `CREATE TABLE`:

- Declarando tras la declaración del nombre y tipo del atributo que referencia a algunos atributos de otra tabla. Esta forma sólo puede usarse cuando la llave es un atributo individual. La declaración es:

  ```
  REFERENCES tabla (atributo)
  ```

  Ejemplo:

  ```
  CREATE TABLE Empleados [
          nombre2 CHAR(10) REFERENCES Almacen (nombre)
          empleados INT
  };
  ```

- Añadiendo a la declaración de tabla una línea indicando el atributo/s que forman la llave exterior. La declaración es

  ```
  FOREING KEY atributos REFERENCES tabla (atributos)
  ```

  Ejemplo:

  ```
  CREATE TABLE Empleados (
          nombre2 CHAR(10)
          empleados INT
          FOREING KEY nombre2 REFERENCES Almacen (nombre)
  );
  ```

Por lo tanto en el ejemplo anterior todo nombre que aparezca en al atributo `nombre2` de `Empleados` ha de aparecer en el atributo `nombre` de `Almacén`. Una excepción es que si de los atributos declarados como llave externa contiene el valor NULL, éste no deberá aparecer en el correspondiente atributo de la tabla referenciada.

Mantenimiento de la integridad referencial.

A la hora de hacer modificaciones de la base de datos tenemos varias alternativas para mantener la integridad. Llamaremos 'R' a la relación que contiene la llave exterior y 'S' a la relación referenciada:

- Política por omisión: Rechazo de las modificaciones violatorias.

  El sistema rechazará toda actualización que viole la restricción de integridad referencial produciendo un error en tiempo de ejecución. Ejemplos de modificaciones violatorias son:

  - Se intenta insertar una nueva tulpla en R cuyo valor en los atributo/s de llave externa no es NULL ni aparece en ninguna tupla de S.
  - Se intenta actualizar alguno de los atributos de llave externa de R a un valor no nulo que no aparece en los correspondientes atributos de S.
  - Se intenta eliminar tuplas de S cuyos atributos referenciados tienen algún correspondiente en R.
  - Se intenta actualizar alguna tupla de S cambiando alguno de sus atributos de forma que tras la actualización los atributos de R pierdan su referencia con S.

  En todos los casos anteriores se rechazarán los cambios dejando inalterada la base de datos.

- Política en cascada.

  En esta política se manejan las eliminaciones o actualizaciones de la relación referenciada. Cuando eliminamos alguna fila de 'S', para mantener la integridad referencial el sistema elimina las correspondientes filas de 'R'. Si se modifica el atributo referenciado de la tabla 'S' también se modifica el correspondiente atributo de 'R' para mantener la integridad referencial.

- Política de asignación de valor Nulo.

  Este método consiste en cambiar el valor declarado como llave exterior de la relación 'R' a 'NULL' cuando se violen las reglas de integridad referencial.

Las políticas anteriores pueden elegirse independientemente en las eliminaciones y actualizaciones, y se formulan en la declaración de la llave exterior. Las declaramos mediante `ON DELETE` (para las actualizaciones de tuplas) o mediante `ON UPDATE` (para las modificaciones de tuplas) seguidas de la selección `SET NULL` o `CASCADE`.

Ejemplo:

```
CREATE TABLE Empleados (
        nombre CHAR(10) REFERENCES Almacen (nombre)
        empleados INT
          ON DELETE SET NULL
          ON UPDATE CASCADE
);
```

B.6. El SQL en un ambinte de programación: SQL embebido

En lenguaje SQL raramente se utiliza sólo en un programa de aplicación. En la práctica la mayoría de las proposiciones de SQL forman parte de un programa más extenso. Normalmente existe un programa más extenso en algún lenguaje anfitrión ordinario como C, pero algunas de sus funciones o alguna de sus proposiciones son en realidad proposiciones SQL.

Una de las posibilidades es escribir un programa en un lenguaje anfitrión, pero con algunas proposiciones especiales de SQL que están incrustadas y que no forman parte del lenguaje anfitrión. El programa entero se envía a un preprocesador, que convierte las proposiciones incrustadas en algo que el lenguaje anfitrión pueda interpretar. Tras esto el programa preprocesado en el lenguaje anfitrión se compila en la forma habitual. Esto es lo que se conoce como SQL Embebido.

B.6.1. El interfaz SQL/lenguaje anfitrión

La trasferencia de información entre la base de datos (a la cual accede el SQL) y el programa del lenguaje anfitrión se realiza a través de variables del lenguaje anfitrión que pueden ser leídas o escritas por proposiciones del SQL. A estas variables se les denomina variables compartidas y se les antepone dos puntos cuando aparecen dentro de una proposición SQL y aparecen sin ellos en las proposiciones del lenguaje anfitrión.

Cuando se quiere insertar proposiciones SQL en el lenguaje anfitrión éstas irán precedidas de las palabras clave EXEC SQL. Estas proposiciones serán procesadas y reemplazadas por llamadas de función adecuadas en el lenguaje anfitrión, sirviéndose de una biblioteca correspondiente de funciones en SQL.

Normalmente el Gestor de Base de Datos ofrece algún mecanismo para la detección de determinadas circunstancias resultantes de ejecuciones de sentencias SQL, como por ejemplo errores. Por ejemplo el motor de base de datos utilizado en este proyecto, PostgreSQL, permite usar la estructura de datos `sqlca`, para analizar la información a cerca de la ejecución de consultas. Entre otras informaciones, en uno de los campos de la estructura se puede encontrar un código identificador de un tipo de error, y en otro, una breve descripción.

B.6.2. La sección `DECLARE`

La declaración de variables compartidas se realiza de la siguiente forma:

```
EXEC SQL BEGIN DECLARE SECTION;
/* lista de variábles */
EXEC SQL END DECLARE SECTION;
```

La forma de las declaraciones de las variables es la que requiere el lenguaje anfitrión. Sólo conviene declarar variables que tengan tipos que puedan manejar tanto el lenguaje anfitrión como el SQL

B.6.3. Inserción de proposiciones

Entre los ejemplos de proposiciones incrustadas se encuentran las proposiciones de eliminación y de actualización y las que crean, modifican o suprimen elementos de un esquema como las tablas y vistas.

Sin embargo las consultas selecionar-de-donde no son incrustables directamente ya que éstas producen como resultado conjuntos de tuplas, mientras que ningún lenguaje anfitrión soporta directamente un tipo de datos de conjuntos. Es por esto por lo que el SQL incrustado ha de utilizar uno de los siguientes mecanismos para conectar el resultado de las consultas a un programa del lenguaje anfitrión.

1. Si la consulta produce una sola tupla, ésta puede ser almacenada en variables compartidas, una para cada atributo. Para ello utilizamos una opción de la proposición seleccionar-de-donde denominada seleccionar un solo renglón.
2. Las consultas que producen más de una tupla pueden ejecutarse si declaramos un cursor para la consulta el cual recorre todas las tuplas en la relación. En su turno cada tupla puede ser capturada e introducida en variables compartidas y después puede ser procesada por el programa del lenguaje anfitrión.

B.6.4. Proposiciones de selección de un solo renglón

La forma de seleccionar un solo renglón es igual a la proposición selecionar-de-donde salvo que después de la cláusula `SELECT` viene la palabra clave `INTO` y una lista de variables las cuales van precedidas de dos puntos. Si el resultado de la selección es una sola tupla entonces el contenido de los atributos se asigna a las variables. Si el resultado es más de una tupla o ninguna entonces no se realiza ninguna asignación y se escribe un código apropiado en la estructura `sqlca`.

Ejemplo: Se desea seleccionar en la relación `Empleados` el número de empleados del almacén denominado PRES y meterlos en la variable num para mostrarlo posteriormente por pantalla. Para ello se utiliza serviremos del siguiente código:

```
void Selectempl (){
     /*declaración de variables compartidas*/
        EXEC SQL BEGIN DECLARE SECTION;
         int  num;
        EXEC SQL BEGIN DECLARE SECTION;
```

```
        EXEC SQL SELECT numero INTO :num   /*proposición de selección*/
        FROM Empleados
        WHERE nombre=PRES;
        printf ("El número de empleados del almacén PRES en \%d n", num);
}
```

B.6.5. Cursores

Como se ha dicho anteriormente, las consultas que puedan tener como resultado más de una fila habrá de usarse un cursor que recorra la relación. Para crear y utilizar un cursor necesitamos las siguientes proposiciones:

1. Una declaración de cursor como la siguiente:

   `ECEC SQL DECLARE` *nombre del cursor* CURSOR FOR *nombre de relación o proposición selecionar-de-donde*

   El cursor recorrerá todas las tuplas de la relación o las tuplas selecionadas.

2. Una proposición `EXEC SQL OPEN` seguida del nombre del cursor, que inicializa el cursor en el primer lugar donde pueda seleccionar una fila.

3. Una o más aplicaciones de proposiciones de captura cuyo propósito es obtener la siguiente tupla de la relación recorrida por el cursor. Cuando se terminan las tuplas, no se devuelve ninguna y se coloca en el campo de `sqlca` un código indica que no se encontró ninguna fila. La proposición de captura `(FETCH)` tiene la siguiente forma:

   `EXEC SQL FETCH FROM` *nombre del cursor* `INTO` *lista de variables compartidas*

   Si hay una tupla que capturar, sus componentes se colocan en esas variables compartidas.

4. La proposición `EXEC SQL CLOSE` seguida del nombre del cursor. Esta proposición cierra al cursor, que ya no recorrerá más tuplas de la relación. El cursor puede ser reiniciado por otra proposición `OPEN`.

B.6.6. Modificaciones por medio del cursor

Cuando un cursor recorre las tuplas de una tabla además de leer y procesar sus valores, también se pueden eliminar o actualizar tuplas. La sintaxis de estas proposiciones `UPDATE` y `DELETE` son las mismas que en el SQL normal exceptuando la cláusula `WHERE`. En el SQL incrustado ésta sólo puede ser `WHERE CURRENT OF` seguida del nombre del cursor.

Ejemplo: en la relación 'Empleados' se desea añadir un empleado al todos los almacenes que tengan por lo menos 10 empleados. Para ello se creará un cursor que vaya examinando el atributo *numero de empleados*.

```
void cambioempl(){}
  EXEC SQL BEGIN DECLARE SECTION;  /*declaración de variables compartidas*/
     int num;
  EXEC SQL END DECLARE SECTION;

        /*declaración del cursor*/
  EXEC SQL DECLARE cursorempl CURSOR FOR SELECT numero FROM Empleados;

        /*apertura del cursor*/
  ESEC SQL OPEN cursorempl;

  While(1){
    EXEC SQL FETCH FROM cursorempl INTO :num;
if (sqlca.sqlcode == 100){
          /* se ha llegado al final del cursor */
          break;
}
       /*captura de la siguiente tupla*/
       if( num>=10){
          EXEC SQL UPDATE Empleados
               SET numero=numero+1
               WHERE CURRENT OF cursorempl;
       }

  }
  EXEC SQL CLOSE cursorempl;   /*cerramos el cursor*/
}
```

Para la confección de este apéndice se ha utilizado la siguiente bibliografía: Introducción a los sistemas de bases de datos [18], Introducción a los sistemas de bases de datos [19], SQL para usuarios y programadores [20].

Apéndice C

Herramientas de postgreSQL

El motor de base de datos postgreSQL incluye dos aplicaciones que se han utilizado para la implementación del benchmark. La primera, el preprocesador ECPG, se utiliza para traducir las sentencias SQL incrustadas en el código del benchmark a funciones de postgreSQL. La segunda, el monitor interactivo PSQL, entre otras cosas, permite verificar las operaciones que el benchmark realiza sobre la base de datos.

A continuación se describen en detalle ambas aplicaciones.

C.1. Preprocesador ECPG

El SQL embebido tiene algunas ventajas sobre otros modos de manejo de consultas SQL. Muchos RDBMS soportan SQL embebido.

Existe un estándar ANSI que describe cómo debería trabajar el lenguaje empotrado. Ecpg se diseñó para cumplir con ese estándar en todo lo posible. Gracias a esto es posible importar programas escritos para otros RDBMS a postgreSQL y de esta manera promover el espíritu del software de libre distribución.

El mecanismo consiste en escribir un programa en C con alguna sentencia SQL. Para declarar variables que puedan ser usadas por las sentencias SQL se necesita declararlas en una sección especial.

Antes de compilar el programa el archivo se pasa por el preprocesador de SQL embebido en C que convierte las sentencias SQL en llamadas a funciones con las variables usadas como parámetros. Después se compila y se enlaza con una librería especial que contiene esas funciones. Esas funciones capturan la información de los argumentos, realizan la consulta SQL usando el interfaz ordinario, y retorna los resultados en los argumentos de salida.

C.1.1. Preprocesado

El preprocesador ecpg reside tras la compilación en el directorio /usr/local/pgsql/bin.

La librería ecpq se llama `libecpg.a` o `libecpg.so`. Además esta librería usa a su vez la librería `libpq` para las comunicaciones con postgreSQL, por lo que habrá que enlazar el programa con las opciónes `-lecpg -lpq`.

Pasos:

- Para el preprocesado del programa se ejecutará el siguiente comando:

  `# ecpg [-d] -o 'fichero de salida' 'fichero de estrada'.`

  La opción '-d' es opcional y permite conectar la opción de depuración.

- Asumiendo que los binarios de postgresSQL están localizados en /usr/local/pgsql, el programa tendrá que ser compilado y enlazado de la siguiente forma:

  ```
  # gcc -g -I /usr/local/pgsql/include -o 'fich. salida' 'fich. entrada'
  -L /usr/local/pgsql/lib -lecpg -lpq
  ```

Manejo de errores

Para poder manejar los errores procedentes de postgreSQL se debe incluir en la sección de librerías del programa la siguiente línea:

```
exec sql include sqlca;
```

Esto define una estructura llamada `sqlca` con el siguiente formato;

```
struct sqlca {
       char sqlcaid[8];
       long sqlabc;
       long sqlcode;
       struct{
             int sqlerrml;
             char sqlerrmc[70];
       } sqlerrm;
       char sqlerrp[8];
       long sqlerrd[6];
       char sqlwarn[8];
       char sqlext[8];
} sqlca;
```

Si ha ocurrido un error en la última sentencia SQL el campo `sqlca.sqlcode` contendrá un valor distinto de 0. Si `sqlca.sqlcode` es menor que 0 eso significa que ha ocurrido un error serio en la base de datos, como que la definición de la base de datos no coincide

con la consulta dada. Si es mayor que 0 significa que ha ocurrido un error normal como que la tabla no contiene la fila solicitada.

El campo `qlca.sqlerrm.sqlerrmc` contiene una cadena que describe el error.

A continuación se muestra una lista con los códigos de algunos de los errores que pueden ocurrir,

- 100. Significa que no se ha encontrado ninguna fila coincidente en la seleccion anterior.
- -402. Significa que no se ha podido conectar con la base de datos.
- -401. Significa que no se ha podido conectar, confirmar o cancelar una transacción.
- -230. Significa que no se ha podido ejecutar la sentencia anterior.

Sentencias SQL embebidas

Todas las sentencias SQL que se empotren en C deben ir precedidas de las palabras:

```
EXEC SQL...
```

La lista de variables compartidas se declara de la siguiente forma:

```
EXEC SQL BEGIN DECLARE SECTION;
      <<Lista de variables>>
EXEC SQL END DECLARE SECTION;
```

Antes de realizar qualquier consulta SQL es necesario conectar con la base de datos mediante la sentencia:

```
EXEC SQL CONNECT TO 'nombre de la base de datos' USER 'nombre de usuario';
```

Una vez hecho esto se podrá escribir cualquier consulta SQL precedida de 'EXEC SQL'.

Cuando se haya terminado de ejecutar sentencias SQL se tendrá que desconectar de la base de datos mediante la sentencia:

```
EXEC SQL DISCONNECT;
```

C.2. Monitor interactivo PSQL

Psql es un front-end en modo texto para postgreSQL. Permite al usuario introducir consultas, enviarlas a postgreSQL, y ver sus resultados. Opcionalmente las entradas

pueden proceder de un archivo. Además psql proporciona un serie de comandos y de características de interprete de comandos que permiten escribir scripts y automatizar una amplia gama de tareas.

Psql es una aplicación común de cliente de postgreSQL. Para conectar con la base de datos el usuario necesita conocer el nombre de la base de datos, el nombre del host, el puerto del servidor y el nombre de usuario con el que se quiere conectar. Si se omite el nombre del host psql conectará mediante el socket de dominio a un servidor en la propia máquina. El número de puerto por defecto se determina en la compilación. El nombre de usuario por defecto es el nombre de usuario de UNIX. Para poder acceder a una base de datos el usuario ha de tener permisos de acceso.

Si la conexión no se ha podido realizar por cualquier motivo psql retorna un mensaje de error y termina.

Un ejemplo de conexión sería:

```
$ psql tpcc postgres
```

C.2.1. Introducción de consultas.

Tras la conexión, psql muestra un prompt con el nombre de la base de datos a la que se ha conectado seguido de la cadena '=>'.un ejemplo de esto es:

```
$ psql tpcc postgres

        Welcome to psql, the postgreSQL interactive terminal.

        Type:   \copyright for distribution terms
                \h for help with SQL commands
                \? for help on internal slash commands
                \g or terminate with semicolon to execute query
                \q to quit

tpcc=>
```

En el prompt el usuario puede introducir comandos o consultas SQL. Normalmente las líneas de entrada son enviadas al backend cuando terminan con ';'. Una consulta no termina con el fin de línea, por lo una consulta puede estar compuesta de varias líneas y sólo se termina con un ';'. Si la consulta enviada no tiene ningún error los resultados se muestran en la pantalla.

C.2.2. Comandos

Cualquier sentencia introducida en el psql y que comience con '\' es un comando de psql. El formato de los comandos es el caracter '\' seguido del nombre del comando y los parámetros.

Algunos de los comandos de psql son;

- `\connect (ó \c) [base de datos] [nombre de usuario]`

  Permite conectar con otra base de datos y/o bajo otro usuario cerrando la conexión previa. Si el nombre de la base de datos es '-' se considera la base de datos actual. Si no se especifica nombre de usuario se considera el actual.

- `\d tabla`

  Muestra todas las columnas de la tabla, sus tipos, y cualquier característica especial de esa columna (ej: valores por defecto). También se listan todos los índices creados para esa tabla.

  El comando \d+ es idéntico pero muestra además cualquier comentario asociado a la columna.

- `\h [comando]`

  Muestra una breve descripción del comando SQL especificado. Si no se especifica ningún comando muestra una lista de todos los comandos sql disponibles.

- `\i 'fichero'`

  Lee sentencias de un fichero y las ejecuta como si fueran introducidas por teclado.

- `\l`

  Lista todas las bases de datos que hay en el sistema así como su propietario. Añadiendo el signo '+' escribe una descripción de cada base de datos.

- `\o {fichero|comando}`

  Escribe el resultado del comando en un fichero. Si no se especifica al nombre del fichero la salida es a 'stdout'.

- `\q`

  Termina el psql.

- `\z`

  Muestra una lista con todas las tablas de la base de datos con sus permisos de acceso.

- `\?`

  Muestra información acerca de los comandos de psql.

Para la realización de este apéndice se han utilizado las siguientes referencias: Manual de Usuario de postgreSQL [28], Manual de Referencia de postgreSQL [27].

Apéndice D

Uso de gnuplot

El visualizador de gráficas gnuplot se utiliza para representar las gráficas de salida de benchmark TPCC-UVA. A continuación se explica el modo de uso de gnuplot para obtener dichas gráficas.

D.1. Trazado de los puntos de un fichero de datos

Gnuplot permite trazar una gráfica definida por los puntos de un fichero de datos. El comando que realiza esta función es el comando `plot 'fichero'`, donde `'fichero'` contiene los puntos de la gráfica a representar.

El formato del fichero debe ser el siguiente:

```
x1 y1
x2 y2
x3 y3
. .
. .
xn yn
```

donde cada par (xn,yn) es un punto a representar.

Las gráficas se pueden representar en distintos formatos añadiendo un modificador al comando anterior.

`plot 'archivo'` Sin modificadores se visualiza la gráfica con puntos.

`plot 'archivo' with lines` Se visualiza la gráfica interpolando mediante líneas cada punto.

`plot 'archivo' with steps` Se visualiza la gráfica mediante escalones.

D.2. Titulado de los ejes

Gnuplot permite poner nombres a los ejes de las gráficas de la siguiente manera:

- Para el eje 'x':
  `>set xlabel '<cadena>'`
- Para el eje 'y':
  `>set ylabel '<cadena>'`

donde 'cadena' es la cadena que se desea colocar en los ejes.

Esta operación tendrá efecto la próxima vez que se ejecute el comando `plot` o `replot`.

D.3. Inserción de etiquetas

Para insertar una etiqueta en un punto determinado de la gráfica se utiliza la expresión:

```
>set label id '<cadena>' at x,y
```

donde:

- `id` es un entero que identifica a la etiqueta.
- `x,y` punto donde se inserta la etiqueta.

Esta operación tendrá efecto la próxima vez que se ejecute el comando `plot` o `replot`.

D.4. Inserción de títulos

Se puede imprimir el título de la gráfica a través del comando:

```
>set title '<cadena>'
```

Cuando se ejecute un `plot` o `replot` aparecerá el título 'cadena'.

D.5. Inserción de líneas en la gráfica

Se puede dibujar una línea en una gráfica utilizando la sentencia siguiente:

```
>set arrow id from x1,y1 to x2,y2
```

donde:

- `id` es un entero que identifica a la línea.
- `x1,y1` punto de inicio de la línea.
- `x2,y2` punto de fin de la línea.

Esta operación tendrá efecto la próxima vez que se ejecute el comando `plot` o `replot`.

D.6. Imprimir una gráfica

Se puede obtener un archivo en formato postscript mediante las siguientes sentencias:

```
>set terminal postscript
>set output 'fichero.ps'
```

donde `fichero.ps` es el fichero de salida.

A partir de este momento cada vez que se haga un `plot` se escribirá en ese fichero.

Si se quiere volver a ver las gráficas por pantalla:

```
>set output
>set terminal x11
```

Una vez que se ha obtenido el fichero `.ps` ya se puede enviar a la impresora.

D.7. Guardar la gráfica en un fichero .eps

Se puede almacenar la gráfica en un fichero `.eps` mediante las siguientes sentencias:

```
>set terminal postscript eps
>set output 'fichero.eps'
```

donde `fichero.eps` es el fichero de salida.

A partir de este momento cada vez que se haga un `plot` o un `replot` se escribirá en ese fichero.

Si se quiere volver a ver las gráficas por pantalla:

```
>set output
>set terminal x11
```

Para la realización de este apéndice se ha utilizado el manual *Gnuplot. An Interactive Ploting Program* [29].

Bibliografía

[1] Estructura y diseño de computadores: interficie, circuitería, programación. David A. Patterson, John L. Hennessy. Ed. Reverte S.A 2000.

[2] Arquitectura de computadores, un enfoque cuantitativo. John L. Hennessy, David A. Patterson. Mc Graw Hill 1993.

[3] The Benchmark Handbook. Jim Grey. Edición electronica disponible en: www.benchmarkresources.com/handbook/introduction.asp. Fecha de acceso: Mayo 2002.

[4] TPC Benchmark TM A. Standard Specification. 10 Noviembre 1989. http://www.tpc.org/tpca/default.asp. Fecha de acceso: Mayo 2002. mayo 2002)

[5] TPC Benchmark TM B. Standard Specification. 23 Agosto 1990. http://www.tpc.org/tpcb/default.asp. Fecha de acceso: Mayo 2002. mayo 2002)

[6] TPC Benchmark TM C. Standard Specification Revision 5.0. 6 Febrero 2001. http://www.tpc.org/tpcb/default.asp. Fecha de acceso: Mayo 2002.

[7] Transaction Processing Performance Council. http://www.tpc.org. Fecha de acceso: Mayo 2002.

[8] Standard Performance Evaluation Corporacion. http://www.specbench.org. Fecha de acceso: Mayo 2002.

[9] The Perfect Club Benchmarks: Effective Performance Evaluation of Supercomputers. Berry, M.D., D. Chen, P. Koss, D. Kuck. CSRD Report No. 827, Center for Supercomputing Research and Development. U. Illinois at Urbana. Nov 1988.

[10] NAS Parallel Benchmark. http://science.nas.nasa.gov/Software/NPB. Fecha de acceso: Mayo 2002.

[11] An Introduccion to High Performance Computing. Stephen Booth, John Fisher, Neil MacDonald, Peter Maccallum, Joel Malard, Alistair Ewing, Elspeth Minty, Alam Simpsom, Stuart Patom, Stephen Breuer. Edinburgh Parallel Computing Centre. The University of Edinburgh. Versión 2.0.

[12] Un protocolo de coherencia para sistemas multicomputador basado en estaciones de trabajo conectadas en bus común. Diego R. Llanos Ferraris. Consejería de Educación y Cultura, Junta de Castilla y León 2002.

[13] Parallel Computer Architecture. A hardware/Software approach. David E. Culler. Jaswinder Pal Singh. Ed. Morgan Kouffmann Publishers Inc. 1999.

[14] Structured Computer Organization. Andrew S. Tanenbaum. Ed. Prentice-Hall international 1999 4º ed.

[15] Advanced Programing in the UNIX Environment. W. Richard Stevens. Ed. Addison - Wesley 1993.

[16] Unix, Programación Avanzada. Francisco Manuel Marquez García. Ed. Ra-Ma. 2º ed. 1996.

[17] Curso de C bajo Unix. Diego Rafael Llanos Ferraris. Ed. Universidad de Valladolid. 1998.

[18] Introducción a los sistemas de bases de datos. Date, C, J. Pearson Educación, Méjico 2001.

[19] Introducción a los sistemas de bases de datos. Jeffrey D. Ullman, Jennifer Widom. Prentice Hall 2000.

[20] SQL para usuarios y programadores. J. Benavidez Abajo, J.M. Olaizola Bartolomé, E. Rivero Cornelio. Ed. Paraninfo 1991.

[21] Linux Edición Especial. David Bandel, Robert Napier. 6ª Edición. Prenctice Hall 1999.

[22] Red Hat Linux. http://www.redhat.com. Fecha de acceso: Mayo 2002.

[23] Using and Porting the GNU Compiler Collection. Versión 2.95. Disponible en: http://gcc.gnu.org/onlinedocs/gcc-2.95.3/gcc.html.

[24] PostgreSQL. http://www.postgresql.org. Fecha de aceso: Mayo 2002.

[25] PostgreSQL 7.1 Administrator's Guide. The postgreSQL Global Development Group. Disponible en: ftp.es.postgresql.org/postgresql/doc/7.1/admin.pdf. Fecha de acceso: Abril 2002.

[26] PostgreSQL 7.1 Programmer's Guide. The postgreSQL Global Development Group. Disponible en: ftp.es.postgresql.org/postgresql/doc/7.1/programmer.pdf. Fecha de acceso: Abril 2002.

[27] PostgreSQL 7.1 Reference Manual. The postgreSQL Global Development Group. Disponible en: ftp.es.postgresql.org/postgresql/doc/7.1/reference.pdf. Fecha de acceso: Abril 2002.

[28] PostgreSQL 7.1 User's Guide. The postgreSQL Global Development Group. Disponible en: ftp.es.postgresql.org/postgresql/doc/7.1/user.pdf. Fecha de acceso: Abril 2002.

[29] Gnuplot. An Interactive Ploting Program. Thomas Williams, Colling Kelley. Versión 3.7. 1998.