Introducción

En los últimos años ha surgido la necesidad de tener acceso y actualizar los datos de gran variedad de bases de datos preexistentes, las cuales se diferencian entre ellas por su entorno de hardware y de software, y por los esquemas en los que se guardan los datos. Un sistema con múltiples bases de datos es un nivel de software que permite que una colección heterogénea de bases de datos de este tipo se trate como una base de datos distribuida homogénea [SK91].

En una *base de datos centralizada* toda la información se encuentra almacenada en un solo reservorio, los datos se acceden e integran con herramientas de la base de datos. Se mantiene el control sobre los datos, tanto en términos de seguridad como de acceso a los mismos. Se puede realizar backup del sistema y recuperación en forma unitaria, y las relaciones entre los datos pueden ser controlados y mantenidos en forma segura [BUR94].

Por otro lado, en una *base de datos distribuída* se almacena la base de datos en varias computadoras. Los medios de comunicación, las redes de alta velocidad y las líneas telefónicas permiten el contacto entre las computadoras de un sistema distribuído. No se comparten memoria ni discos. Las computadoras pueden variar en tamaño y función, pudiendo abarcar desde las estaciones de trabajo a los grandes sistemas.

Las computadoras pueden denominarse como nodos o emplazamientos a la hora de enfatizar la distribución física.

Las principales características de las bases de datos distribuídas son las siguientes:

- Normalmente se encuentran en varios lugares geográficos distintos, se administran de forma separada y poseen una interconexión más lenta.
- En un sistema distribuído se dan dos tipos de transacciones: las transacciones locales, que acceden a los datos de un único nodo, generalmente en el que se inició la transacción, y las transacciones globales, que acceden a los datos situados en uno o más nodos diferentes a aquel en el que se inició la transacción.

En síntesis, los sistemas de bases de datos distribuídos consisten en nodos poco acoplados que no comparten componentes físicos. Además, cada nodo puede tener un grado significativo de independencia mutua.

Existen varias razones para construir sistemas distribuídos de bases de datos, incluyendo el compartimiento de datos, fiabilidad y disponibilidad, y la aceleración del procesamiento de consultas. Pero estas ventajas están acompañadas por varios

inconvenientes tales como un mayor costo en el desarrollo de software, una mayor posibilidad de fallos y un incremento de la sobrecarga de procesamiento. Además el mayor inconveniente es la complejidad añadida para asegurar la correcta coordinación entre los nodos.

Las razones fundamentales para la introducción de *bases de datos distribuídas* son básicamente cuatro:

- Necesidad de bases de datos especializadas.
- Movimientos corporativos "rigthsizing" (adecuados, correctos).
- Adquisiciones corporativas.
- Mejorar la performance y confiabilidad, a través de la disponibilidad de los datos.

Todo esto lleva a estudiar en forma detallada los *sistemas de bases de datos distribuídos* de manera de poder alcanzar, al menos en la etapa de diseño, un modelo de datos que al momento de implementarse, sea lo suficientemente eficiente como para cubrir las expectativas de performance mínimas requeridas en una corporación.

Motivación

Los avances en la tecnología llevan a mayor velocidad en acceso a datos. A esto se suma una mayor demanda de los usuarios, no solo en velocidad, sino que en funcionalidad, servicios, flexibilidad y performance.

Sumado a los avances tecnológicos, la introducción de facilidades para comunicación entre redes motivan la distribución de los datos. Las ventajas que se obtienen son rápidamente observables: se provee evolución en sistemas o cambios en los requerimientos del uso, se permite autonomía local, la arquitectura del sistema es simple, tolerante a fallas y flexible, y se mejora considerablemente la performance a nivel local. Pero también con este cambio aparecen dificultades, principal atención de los investigadores:

- Asegurar el acceso a otros sitios y procesamiento eficiente.
- Transformación de datos e integración entre nodos / sitios.
- Distribución de los datos en la red de manera óptima.
- Control de acceso a los datos de la red.
- Soporte de recuperación ante fallas en forma eficiente y segura.

- Asegurar que los sistemas locales y globales sean un modelo del mundo real, considerando evitar interferencias destructivas entre diferentes transacciones contra el sistema.

El tercer punto motiva el desarrollo de este trabajo de grado. La distribución de los datos en forma eficiente en los sitios de la red distribuída es una tarea importante en el diseño de la base de datos distribuída (en adelante BDD), de manera de asegurar performance y accesibilidad a los datos. Un buen diseño minimiza los problemas en la etapa de implementación. Todo esto trata de proveerse con la herramienta propuesta, denominada *SimReplica*, basándose en la distribución de los datos (réplicas) y en el porcentaje de operaciones que soportará el modelo de datos real que se está evaluando.

Objetivos generales

Crear una herramienta amigable que permita diseñar la base de datos distribuída para modelos de datos reales, donde el análisis estará basado en la replicación de los datos y en el porcentaje de operaciones (altas, bajas, modificaciones y consultas) que afectarán los datos del modelo.

Objetivos específicos

Los objetivo específicos de este trabajo de grado son los siguientes:

1. Implementar una herramienta que permita ayudar el diseño de bases de datos distribuídas.
2. Implementar un esquema de replicación de datos, basándose en la forma de propagación y actualización de los mismos.
3. Analizar los resultados obtenidos con la herramienta desarrollada.
4. Evaluar algoritmos alternativos para la propagación y actualización de datos en la BDD en forma teórica.

Desarrollo del proyecto

Para alcanzar los objetivos, se realizaron los siguientes pasos:

1. Se ampliaron conocimientos respecto del campo de procesamiento distribuído y bases de datos distribuídas, reforzando el entorno teórico sobre el cual se desarrolló el proyecto.
2. Se estudiaron las diferentes técnicas de actualización de replicación, evaluando ventajas y desventajas de cada uno de ellos.

3. Se diseñó e implementó el algoritmo de replicación elegido utilizando como plataforma de desarrollo JAVA.
4. Se diseñó e implementó la interfaz visual de la herramienta, eligiendo para ello un lenguaje de programación que permita un desarrollo de interfaces con el usuario amigables, y que además pueda integrarse con lo desarrollado en JAVA.
5. Se diagramaron casos de estudio, que fueron posteriormente evaluados con la herramienta, analizando cuidadosamente los resultados, a fin de determinar el grado de exactitud y veracidad de los algoritmos implementados.

Organización del contenido

El capítulo 1 introduce los conceptos fundamentales de la replicación en Bases de Datos Distribuídas, conceptos necesarios e indispensables para ubicar la naturaleza del proyecto.

El capítulo 2 profundiza en los conceptos sobre la replicación de datos, detallando métodos de propagación de actualizaciones, métodos de regulación de actualizaciones y finalmente la combinación entre ambos, que lleva a los protocolos de replicación.

El capítulo 3 hace una breve referencia a las herramientas usadas en la implementación del simulador, justificando su uso.

El capítulo 4 desarrolla exhaustivamente el trabajo realizado, definiendo y acotando el problema estudiado.

El capítulo 5 presenta los resultados obtenidos en la simulación a partir de un caso de prueba concreto.

El capítulo 6 detalla las conclusiones del trabajo.

Finalmente, se agrega a modo de apéndice los casos de prueba evaluados, a fin de justificar los resultados obtenidos.

Capítulo 1: Bases de datos distribuídas

Definiciones

Una **base de datos distribuída (BDD)** es una colección de bases de datos lógicamente interrelacionadas distribuídas sobre una red de computadoras. El punto importante de los sistemas distribuídos de datos es que el usuario ve esta colección como si fuese una única entidad.

Un **sistema gerenciador de bases de datos distribuídas (DBMS)** es definido como el software que permite el manejo de las bases de datos distribuídas y que hace que la distribución sea transparente a los usuarios.

Una BDD ideal debería cumplir con las siguientes especificaciones [BUR94]:

- **Autonomía local:** todos los datos de la red distribuída pueden ser manipulados localmente.
- **No dependencia de un sitio central:** Idealmente, todos los sitios son igualmente remotos, y ninguno tiene autoridad sobre otros nodos. Cada sitio tiene su propio diccionario de datos y su esquema de seguridad.
- **Operación continua:** mientras un sitio tiene su propia identidad y control, sus funciones unificadas como parte de una federación deben estar siempre disponibles para otros nodos.
- **Independencia de locación:** los usuarios finales no deben necesariamente saber donde están los datos que reclama.
- **Independencia de fragmentación:** se refiere a la habilidad de los usuarios finales de almacenar información lógicamente relacionada en lugares físicos diferentes.
- **Independencia de replicación:** es la habilidad de la base de datos de crear copias del master en sitios remotos.
- **Procesamiento distribuído de consultas:** es la habilidad de ejecutar consultas contra más de una base de datos.
- **Gerenciamiento distribuído de transacciones:** se refiere a un sistema que puede manejar una actualización, inserción o borrado a muchas bases de datos a partir de una consulta simple.

- **Independencia de hardware:** se refiere a la habilidad de una consulta de consultar y actualizar información a pesar de la plataforma de hardware en la cual residen los datos.
- **Independencia del sistema operativo.**
- **Independencia de la red:** los protocolos de la red no deben ser una característica de la base de datos.
- **Independencia de la base de datos:** debe ser posible recuperar y actualizar información de bases de datos distintas con arquitecturas diferentes.

Por otro lado, las BDD traen consigo una serie de desventajas:

- **Falta de Experiencia:** Los sistemas de BDD de propósito general no son aún muy usados. Actualmente existen prototipos o sistemas que funcionan solamente para una aplicación. Las soluciones que han sido propuestas para varios problemas reales no han sido aún testeados en sistemas realmente operativos.
- **Complejidad:** Los problemas en los sistemas de BDD son inherentemente más complejos que los de bases de datos centralizadas. Se encuentran los problemas de BD centralizadas sumados a los problemas que surgen de la distribución de los datos.
- **Costo:** Los sistemas distribuídos requieren de hardware adicional, lo cual incrementa los costos. La tendencia a la baja en los costos de hardware, resta importancia a este punto, pero se introducen los costos relacionados con el software para la administración de estos sistemas.
- **Distribución del control:** La distribución crea problemas de sincronización y coordinación.
- **Seguridad:** Uno de los mayores beneficios de los sistemas de BD centralizados es el control que proveen respecto del acceso a los datos. En un sistema de BDD una red es el medio de acceso a los datos, y ésta tiene sus propios requerimientos de seguridad, los cuales deben ser además combinados con los mecanismos de seguridad de acceso a los datos.
- **Dificultad de cambios:** Es difícil cambiar de una base de datos centralizada funcionando a una BDD con todos los requerimientos que ésta necesita.

Arquitectura de bases de datos distribuídas

La mayoría de los DBMS's disponibles se basan en la arquitectura ANSI-SPARC, que divide al sistema en 3 niveles (Fig. 1.1):

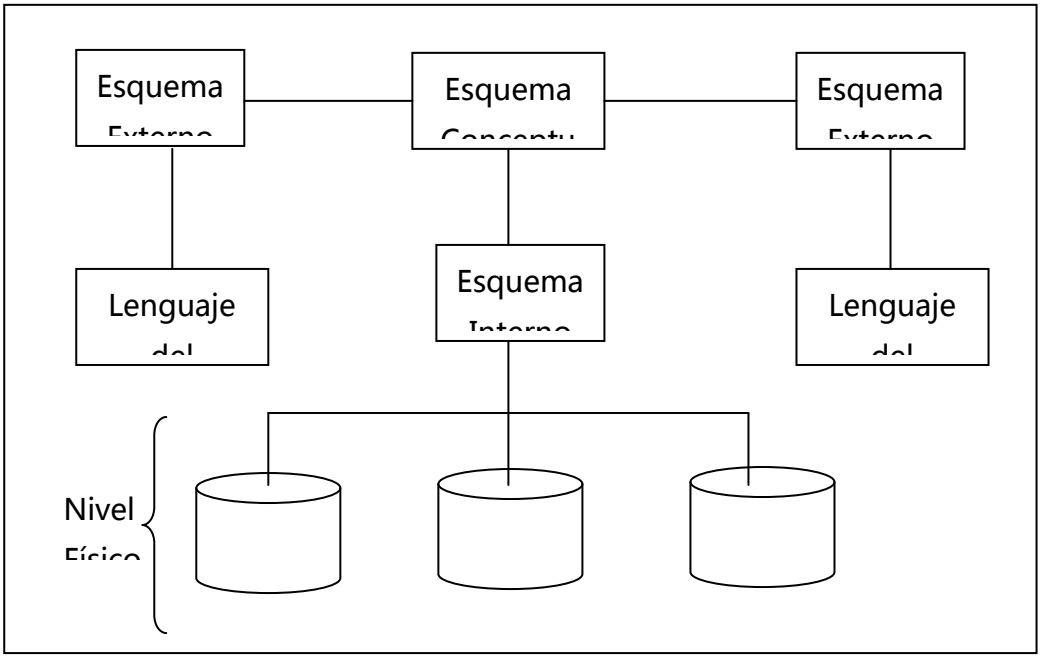


Fig. 1.1: Arquitectura de BDD

- **Conceptual:** representa como los usuarios ven los datos de la base de datos. También es llamada VISTA LOGICA GLOBAL. La generación del esquema conceptual es el primer paso en el diseño de la base de datos.
- **Externo:** Provee una ventana para la vista conceptual que permite a los usuarios ver sólo los datos de su interés. Los programas del usuario para acceso a los datos utilizan esta vista.
- **Interno:** El esquema conceptual se mapea en un esquema interno, el cual representa la descripción de los datos de la base de datos en bajo nivel. Provee una interfaz con el sistema operativo (responsable de acceder a la base de datos). El nivel interno tiene que ver con:
  - ✓ Que ítems de datos están indexados
  - ✓ Técnicas de organización de archivos a usar
  - ✓ Como se alojan los datos en el disco para mayor performance

Componentes de un DBMS

- **Catálogo o Diccionario de Datos:** con este interactúan todos los demás componentes del sistema. Contiene todos los metadatos del sistema (descripción de

todos los ítems de datos, con su tipo, longitud, valores posibles, etc., permisos de acceso, vistas, etc.). Los tres niveles de la arquitectura de un DBMS interactúan entre ellos a través del catálogo.

- **Procesador de Consultas:** es el responsable de aceptar comandos de los usuarios finales para almacenar, recuperar y actualizar datos de la base de datos. Para resolver las consultas usa el catálogo, interpreta los pedidos y los traduce a pedidos de acceso a los datos físicos. Definiendo una transacción como el proceso de recuperar y actualizar datos, un DBMS convencional debe soportar varias transacciones activas a la vez.
- **Gerenciador de Transacciones:** Asegura que las transacciones no interfieran entre ellas y corrompan la base de datos.
- **Gerenciador de recuperaciones:** minimiza el efecto de fallas en la base de datos. El objetivo es restaurar la base de datos a un estado conocido y consistente.

Áreas de problemas en BDD

Existen varios problemas técnicos que deben resolverse antes de poder obtener los beneficios potenciales de los DBMS. Algunos de estos problemas son:

- **Diseño de la base de datos distribuída:** las aplicaciones que ejecutan contra la base de datos, ¿en qué sitios del sistema distribuído deben buscar los datos? Existen dos alternativas básicas para alojar los datos: *particionamiento* o *replicación* (más adelante se desarrolla este tema con más detalle).
- **Procesamiento distribuído de consultas:** el procesamiento de consultas trata con algoritmos que analizan las mismas y las convierten en una serie de operaciones que puedan manipular datos. El problema es como decidir la estrategia para que cada consulta en la red distribuída sea de la forma más eficiente. Los factores a ser considerados son la distribución de los datos, los costos de comunicación, y la cantidad de información disponible localmente. El objetivo es optimizar en aquellos lugares donde el paralelismo sea inherente, y así asegurar performance en la ejecución de las transacciones.
- **Gerenciamiento del directorio distribuído:** un directorio contiene la información de todos los items de datos en la base de datos. El problema se plantea en como administrar ese directorio: global a toda la base de datos, o local en cada sitio, centralizado o distribuído, con una sola copia, o con múltiples copias.

- **Control de concurrencia distribuído:** involucra la sincronización de los accesos a la base de datos de manera que se mantenga la integridad de la misma. En un entorno distribuído se requiere consistencia mutua: todos los valores de todas las copias deben converger al mismo valor. Las soluciones son numerosas, pero se pueden mencionar dos clases: *pesimistas*, donde debe sincronizarse la ejecución de los pedidos del usuario antes de iniciar la ejecución, y *optimistas*, se ejecutan los pedidos y luego se chequea si la ejecución compromete la consistencia de la base de datos. Para ambas aproximaciones se utilizan dos primitivas: *locking*, que se basa en la exclusión mutua de acceso a los ítems de datos, y *timestamping*, donde las transacciones se ejecutan en algún orden. Existen variaciones híbridas a estos esquemas que tratan de combinar los dos mecanismos básicos.
- **Gerenciamiento del deadlock distribuído:** la competencia entre usuarios por el acceso a los datos puede resultar en deadlock si el mecanismo de sincronización está basado en el locking. A las bases de datos también pueden aplicarse alternativas de prevención, detección y recuperación.
- **Confiabilidad de DBMS distribuídos:** una de las ventajas de los sistemas distribuídos es la confiabilidad y disponibilidad. Pero estas dos características no aparecen en forma automática. Es tan importante proveer mecanismos para asegurar la consistencia como para recuperar al sistema de fallas: cuando un problema ocurre y algún sitio se torna inoperable o inaccesible, la base de datos en los sitios operativos debe permanecer consistente y actualizada. Esta situación es especialmente difícil de mantener cuando el problema hace que algunos de los nodos queden inaccesibles por problemas de comunicación o particiones en la red.
- **Soporte del sistema operativo:** la actual implementación de los sistemas de bases de datos distribuídas en la capa superior de los sistemas operativos convencionales no es del todo apropiada, por lo que se afecta la performance del sistema. En entornos distribuídos, aparte de tratar con temas como gerenciamiento de memoria, métodos de acceso a archivos, recuperación ante fallas y gerenciamiento de procesos, los sistemas operativos deben tratar con las diferentes capas del software correspondiente a la red.
- **Bases de datos heterogéneas:** Cuando no existe homogeneidad entre las diferentes bases de datos en los diferentes sitios en cuanto a la estructura lógica de los datos (modelo de datos) o en cuanto a los mecanismos de acceso a los mismos (lenguaje

de datos), es necesario proveer un mecanismo de traducción entre los diferentes sistemas.

La relación entre los diferentes problemas mencionados puede resumirse en el cuadro de la Fig. 1.2.

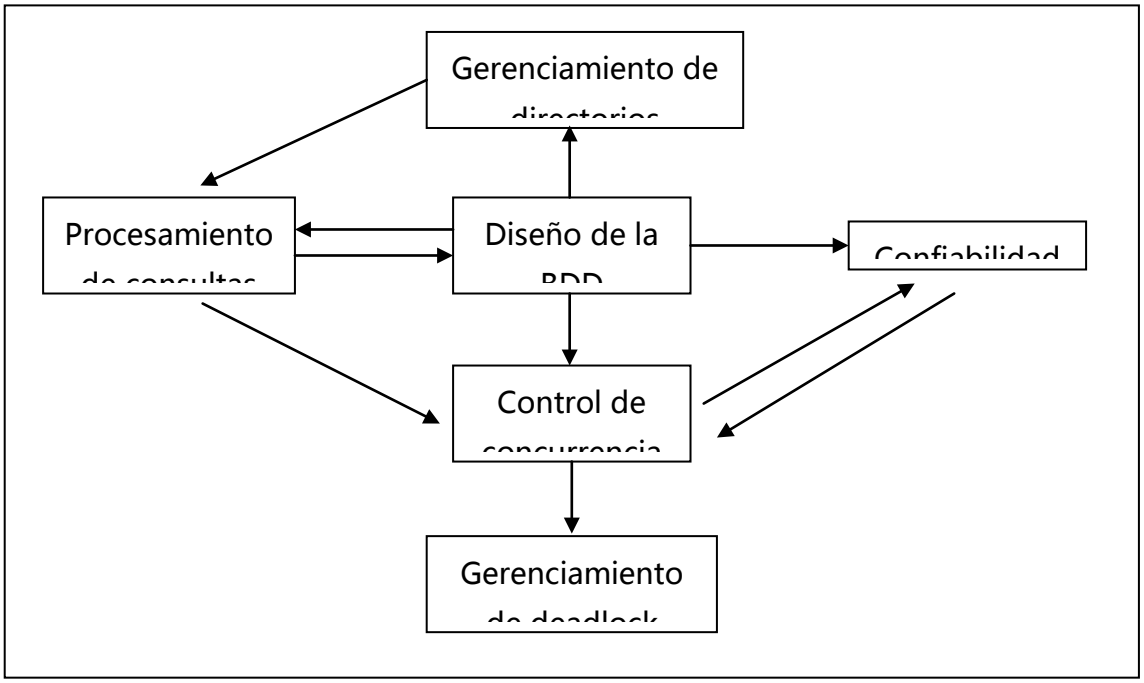


Fig. 1.2: Areas de problemas en BDD

Implementación de un DBMS

Existen muchas alternativas de implementación de DBMS. La más directa es *arquitecturas cliente / servidor*, donde múltiples clientes acceden a una única base de datos. Arquitecturas como *múltiples-clientes / múltiples servidores* son más flexibles porque la base de datos es distribuída a través de varios servidores. Cada máquina cliente tiene un server local a los cuales los usuarios locales acceden directamente. La comunicación entre servers para ejecutar consultas de usuario es transparente a los usuarios.

Un DBMS realmente distribuído no distingue entre máquinas clientes y servidores. Idealmente, cada sitio puede cumplir el rol de cliente o servidor, de acuerdo a la circunstancia. Estas arquitecturas son llamadas *peer-to-peer*, y requieren protocolos sofisticados para manejar los datos distribuídos a través de múltiples sitios.

La base de datos es físicamente distribuída a través de los sitios *fragmentando* y *replicando* los datos.

Dado una base de datos relacional, la fragmentación subdivide cada relación en particiones verticales u horizontales. La fragmentación es adecuada cuando hay que ubicar los datos en los sitios en que son más solicitados. Esto disminuye los costos de transmisión de datos a través de la red y reduce el tamaño de las relaciones involucradas en las consultas de usuario.

Basados en los patrones de acceso, cada fragmento puede ser además replicado. Esto es preferible cuando los mismos datos son requeridos en diferentes sitios.

DBMSs paralelos y distribuídos proveen la misma funcionalidad que DBMSs centralizados, excepto porque los datos se manejan distribuídos a través de sitios en una red de computadoras o nodos en un sistema multiprocesador. Esta funcionalidad es provista en forma transparente: los usuarios no ven la distribución de los datos, la fragmentación o replicación de los mismos.

Particionamiento Vs. Replicación

La replicación de tablas implica la duplicación de las mismas en cada sitio de la red distribuída [BOB96]. Cada tabla puede ser replicada en forma total (Fig. 1.3).

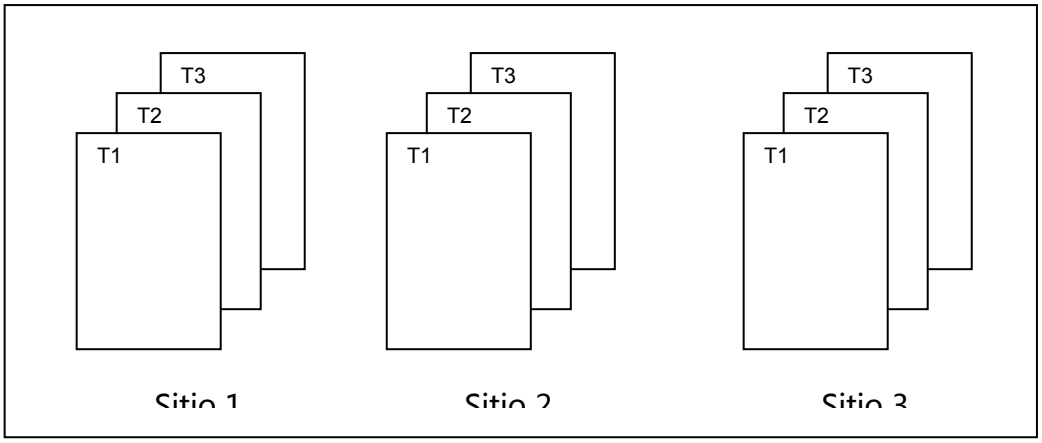


Fig. 1.3: Replicación de Tablas

Alternativamente, algunos esquemas requieren que solamente porciones de las tablas sean duplicadas en cada sitio (Fig. 1.4)

Aunque estos métodos garantizan la seguridad de los datos a través de copias redundantes, son caros de mantener debido al costo de la ejecución sincronizada de inserciones, modificaciones y borrados en cada una de las copias.

Un método alternativo a la replicación es el *particionamiento o fragmentación* de las tablas ya sea a través de sus columnas o filas. Lo primero es llamado *fragmentación horizontal*, mientras que la segunda, *fragmentación vertical* (Fig. 1.5a y 1.5b).

La fragmentación híbrida de tablas es la tercera alternativa de particionamiento. Combina los métodos de fragmentación horizontal y vertical.

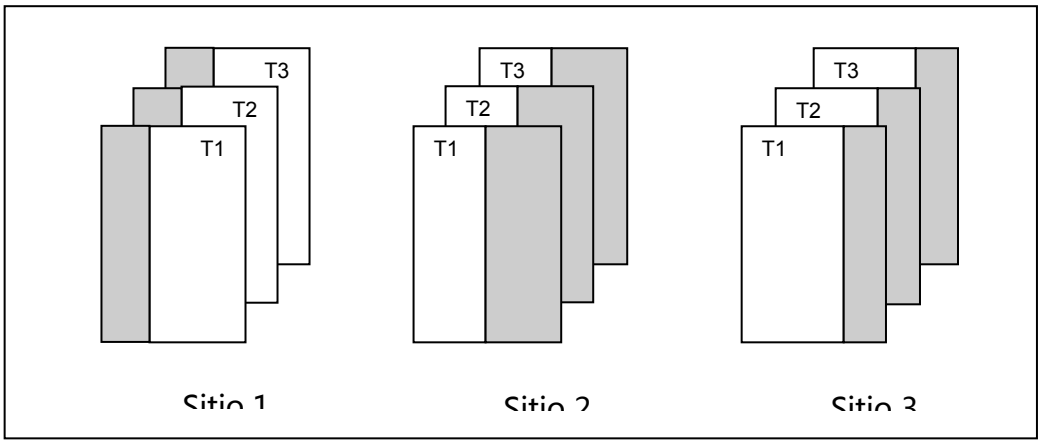


Fig. 1.4: Replicación Parcial de tablas

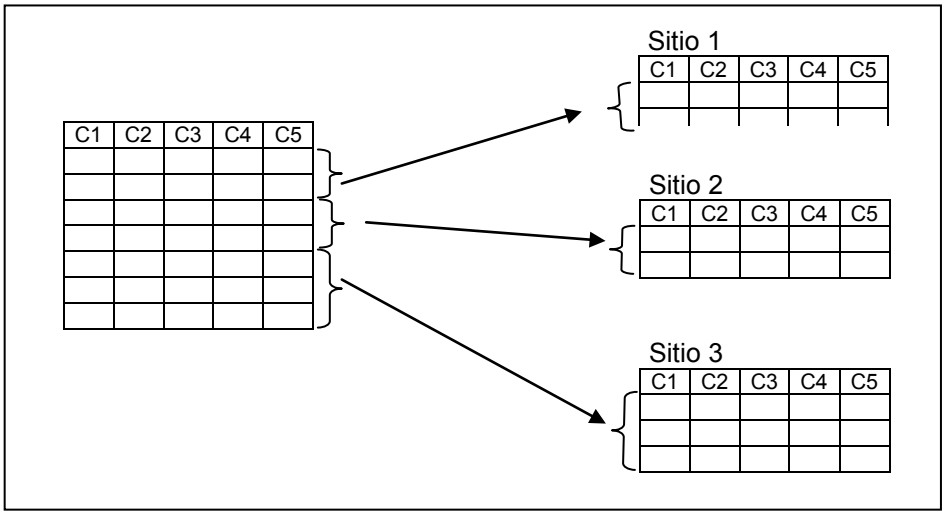


Fig.1.5a: Fragmentación Horizontal

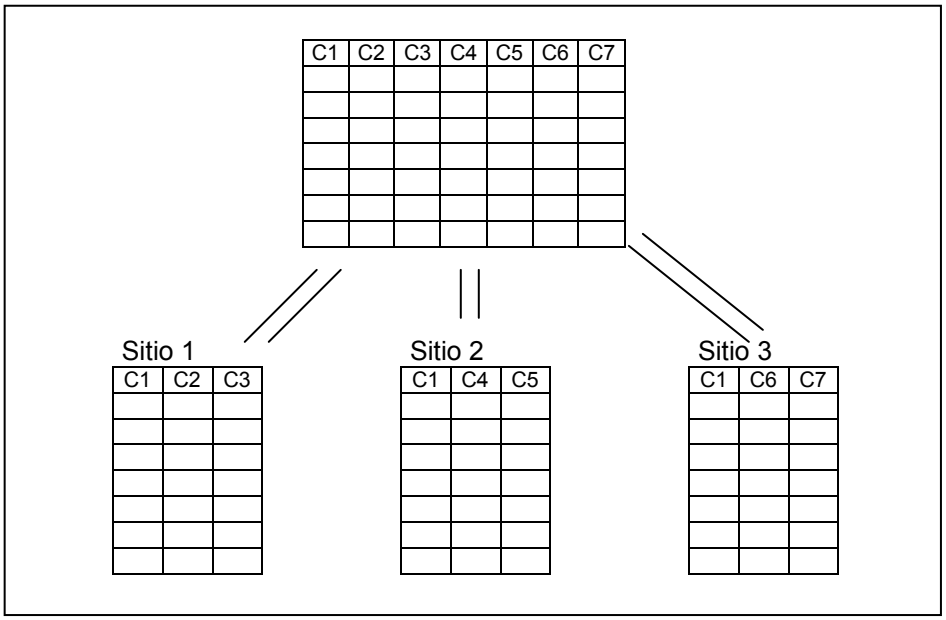


Fig. 1.5b: Fragmentación Vertical

Alocación

La alocación de los recursos a través de los nodos de una red de computadoras es un problema que ha sido estudiado exhaustivamente. Sin embargo, la mayoría de esta investigación no se basa en el problema del diseño de la base de datos, sino que lo hace en la alocación de los archivos individuales dentro de la red de computadoras. [OVA91]

Asumamos que existe un conjunto de fragmentos $F = \{F_1, F_2, ..., F_n\}$ y la red consiste de un conjunto de sitios $S = \{ S_1, S_2, ..., S_m\}$ en los cuales se ejecutan un conjunto de aplicaciones $Q = \{Q_1, Q_2, ..., Q_q\}$. El problema de alocación consiste en encontrar la distribución óptima de F en S.

La optimalidad puede ser definida en función de dos medidas:

1) Costo Mínimo: La función de costo consiste en el costo de almacenar cada F_i en un sitio S_j, el costo de consultar F_i en S_j, el costo de actualizar F_i en S_j, y el costo de la comunicación de los datos. El problema de alocación trata entonces de minimizar la función de costo combinada.
2) Performance: La estrategia de alocación es diseñada para mantener la métrica de la performance, dentro de los que se encuentran minimizar el tiempo de respuesta, y maximizar el "throughput" del sistema en cada sitio.

Se debería buscar un esquema de alocación que responda consultas en un tiempo mínimo mientras que el tiempo de procesamiento también sea mínimo. Aún no se han diseñado esquemas de alocación que respondan a estos dos parámetros en forma

conjunta: siempre se ha focalizado en uno de estos dos problemas. La razón de no estudiar esquemas de alocación teniendo en cuenta las dos métricas anteriores es la complejidad.

Para resolver el problema de alocación deben tenerse en cuenta los siguientes puntos:

1) No pueden tratarse los fragmentos como archivos individuales que pueden alocarse uno por vez, como si estuviesen aislados. El posicionamiento de un fragmento tiene usualmente un impacto en las decisiones de posicionamiento de otros fragmentos que son accedidos en forma conjunta con el primero. En consecuencia debe tenerse en cuenta la relación entre fragmentos para minimizar el costo de acceso a los datos.
2) El acceso a los datos a través de aplicaciones es modelado en forma simple. Un pedido de usuario en un sitio debe hacer que todos los datos para responderlo sean transferidos a ese sitio. En sistemas de BDD el acceso a los datos es más complicado que el acceso a archivos remotos. En consecuencia, el alocamiento y el procesamiento de consultas debe ser modelado apropiadamente.
3) Estos modelos no tienen en cuenta el costo de integración de los datos. En consecuencia, dos fragmentos que están relacionados con la misma regla de integridad en dos sitios diferentes son costosos.
4) Similarmente, debe considerarse el costo de implementar mecanismos de control de concurrencia cuando se cuenta con datos replicados.

No existen modelos de manera que para un conjunto de fragmentos de entrada pueda producir un esquema de alocación óptimo sujeto a los tipos de restricciones discutidos. Los modelos desarrollados hasta ahora se basan en una serie de suposiciones que simplifican el problema y que son aplicables a formulaciones específicas.

Para resolver el problema de la alocación se describe un número de heurísticas posibles que pueden ser usadas en esta tarea:

Información de la base de datos: Para ejecutar la fragmentación horizontal, se define la "selectividad" de términos mínimos: la selectividad de un fragmento F_j con respecto a una consulta q_i es la cantidad de tuplas de F_j que necesitan ser accedidas para procesar q_i, y se denotará $sel_i(F_j)$.

Otra pieza de información necesaria en los fragmentos de la base de datos es su tamaño. El tamaño de un fragmento está dado por:

$$size(F_j) = card(F_j) * length(F_j)$$

donde *length*(Fj) es la longitud (en bytes) de una tupla de fragmentos de F_j.

Información de la aplicación: La mayoría de las aplicaciones ya se encuentran compiladas durante la actividad de fragmentación, pero algunas son requeridas por el modelo de alocación. Dos medidas importantes son el número de lecturas que la consulta q_i realiza sobre el fragmento F_j durante la ejecución (denotado como RR_{ij}) y la cantidad de accesos para actualización (UR_{ij}). Esto puede, por ejemplo, contar la cantidad de accesos a un bloque que requiere la consulta.

También es necesario definir dos matrices UM y RM, con elementos u_{ij} y r_{ij} respectivamente:

$$u_{ij} = \begin{cases} 1 & \text{si } q_i \text{ actualiza el fragmento } F_j \\ 0 & \text{en caso contrario} \end{cases}$$

$$r_{ij} = \begin{cases} 1 & \text{si } q_i \text{ se recupera del fragmento } F_j \\ 0 & \text{en caso contrario} \end{cases}$$

También se define un vector O de valores o(i), donde o(i) especifica el sitio que origina q_i. Finalmente, para definir el tiempo de respuesta, debe ser especificado en función del máximo tiempo de respuesta permitido por cada aplicación.

Información del sitio: Para cada sitio es necesario saber sobre su capacidad de almacenamiento y procesamiento. Estos valores pueden ser obtenidos en términos de funciones elaboradas o a través de simples estimaciones. El costo unitario de almacenamiento de datos en el sitio S_k será denotado por USC_k. LPC_k será el costo de procesar una unidad de trabajo en el sitio S_k. La unidad de trabajo debe ser idéntica a la usada en las medidas RR y UR.

Información de la red: Se asume la existencia de una red simple donde el costo de la comunicación se define en términos de un frame de datos. Entonces, g_{ij} denota el costo de comunicación por frame entre los sitios S_i y S_j. Para poder calcular el número de mensajes, se usa $fsize$ como el tamaño en bytes de un frame.

Modelo de alocación

El modelo de alocación debe minimizar el costo total de procesamiento y almacenamiento a la vez que hay que considerar ciertas restricciones en los tiempos de respuesta. El modelo usado tendrá la siguiente forma:

Min(Costo Total)

sujeto a:

- Restricciones en cuanto al tiempo de respuesta
- Restricciones de almacenamiento
- Restricciones de procesamiento

La variable de decisión es x_{ij}, que se define como sigue:

$$x_{ij} = \begin{cases} 1 & \text{si el fragmento } F_i \text{ es almacenado en el sitio } S_j \\ 0 & \text{en caso contrario} \end{cases}$$

Costo Total: La función de costo total tiene dos componentes: procesamiento de consultas y almacenamiento. Puede ser expresada de la siguiente forma:

$$TCO = \sum_{\forall q_i \in Q} QPC_i + \sum_{\forall S_k \in S} \sum_{\forall F_j \in F} STC_{jk}$$

donde:

QPC_i es el costo de procesamiento de la aplicación q_i

STC_{jk} es el costo de almacenar el fragmento F_j en el sitio S_k

El costo de almacenamiento está dado por:

$$STC_{jk} = USC_k * size(F_j) * x_{jk}$$

y las dos sumas alcanzan el costo total de almacenamiento en todos los sitios para todos los fragmentos.

El costo de procesamiento es más difícil de especificar. Se hará en términos de costos del procesamiento (PC) y costo de transmisión (TC). En consecuencia, el costo de procesamiento (QPC) para la aplicación q_i será:

$$QPC_i = PC_i + TC_i$$

A su vez, el componente de procesamiento consiste de tres factores: costo de acceso (AC), costo de integridad (IE) y costo de control de concurrencia (CC):

$$PC_i = AC_i + IE_i + CC_i$$

La especificación detallada de estos factores depende de los algoritmos usados. Detallando AC,

$$AC_i = \sum_{\forall S_k \in S} \sum_{\forall F_j \in F} (u_{ij} * UR_{ij} + r_{ij} * RR_{ij}) * x_{jk} * LPC_k$$

Los dos primeros términos calculan el número de accesos de la consulta q_i al fragmento F_j. Cabe destacar que (UR_{ij} + RR_{ij}) provee el número total de actualizaciones y accesos para recuperación. Se asume que el costo local de procesamiento es idéntico. La sumatoria da el número total de accesos para todos los fragmentos referenciados por q_i. La multiplicación por LPC_k da el costo de acceso al sitio S_k. Se usa x_{jk} para seleccionar solo aquellos costos para sitios en que los valores están almacenados.

La función de costo de acceso asume que el procesamiento de una consulta se puede descomponer en una serie de subconsultas, donde cada una trabaja en un fragmento almacenado en un sitio, seguido de la transmisión de los resultados al sitio que originó la consulta.

El costo de integridad puede ser especificado más como un componente de procesamiento, excepto que el costo de procesamiento local probablemente cambie para reflejar el costo de integridad real.

La función de costo de transmisión puede ser formulada sobre las líneas de la función de costo de acceso. Sin embargo, el overhead en la transmisión de los datos para actualizaciones y pedidos de recuperación de datos son muy diferentes. En consultas de actualización es necesario informar todos los sitios en que existen réplicas, mientras que en consultas de recuperación de datos es suficiente con acceder solamente a una de las copias. Por otro lado, en las actualizaciones no hay datos de respuesta al sitio que originó la consulta más que un mensaje de confirmación, mientras que en las consultas los datos que retornan al sitio original pueden ser representativos.

La componente de actualización de la función de transmisión es:

$$TCU_i = \sum_{\forall S_k \in S} \sum_{\forall F_j \in F} u_{ij} * x_{jk} * g_{o(i),k} + \sum_{\forall S_k \in S} \sum_{\forall F_j \in F} u_{ij} * x_{jk} * g_{k,o(i)}$$

El primer término es para mandar el mensaje de actualización desde el sitio origen o(i) de q_i a todos los fragmentos replicados que necesitan ser actualizados. El segundo término es para la confirmación.

El costo de recuperación puede ser especificado como:

$$TCR_i = \sum_{\forall F_j \in F} \min_{S_k \in S} (u_{ij} * x_{jk} * g_{o(i),k} + r_{ij} * x_{jk} * \frac{sel_i(F_j)}{fsize} * g_{k,o(i)})$$

El primer término en TCR representa el costo de transmitir la respuesta a aquellos sitios que tienen copias de los fragmentos que necesitan ser accedidos. El segundo término registra la transmisión de los resultados desde estos sitios hasta el sitio origen.

La función de costo de transmisión para una consulta q_i puede ser definida como:

$$TC_i = TCU_i + TCR_i$$

Las funciones de restricción pueden ser definidas en forma similar. La restricción de tiempo de respuesta puede ser especificada de siguiente forma:

$$\textit{Tiempo de ejecución de } q_i \leq \textit{tiempo de respuesta máximo de } q_i, \forall q_i \in Q$$

La restricción de almacenamiento es:

$$\sum_{\forall F_j \in F} STC_{jk} \leq \textit{la capacidad de almacenamiento en el sitio } S_k, \forall S_k \in S$$

Finalmente, la restricción de procesamiento se define como:

$$\sum_{\forall q_i \in Q} \textit{procesamiento de } q_i \textit{ en el sitio } S_k \leq \textit{la capacidad de procesamiento de } S_k, \forall S_k \in S$$

Conclusiones

Todo el detalle referente a la alocación de fragmentos tiene como objetivo destacar que en el diseño de una BDD también debe tenerse en cuenta e incluirse esta tarea. Aunque las investigaciones sobre el tema no han desarrollado estándares que puedan ser aplicados en forma directa a la solución de cualquier problema, es importante destacar los aspectos a tener en cuenta para la resolución de esta tarea. Por otro lado, constituye un aspecto importante en el entorno teórico del tema propuesto en este trabajo de grado.

Capítulo 2: Replicación

Definiciones

Un *objeto replicado* es un objeto de datos tipado cuyo estado es almacenado en forma redundante en muchas localidades para aumentar la disponibilidad, la accesibilidad y la performance, entre otros factores. [HER87]

Un *método o protocolo de replicación* es un algoritmo para administrar las componentes distribuídas de un objeto, de manera que su comportamiento sea equivalente al de un único objeto en un solo sitio. Esta propiedad es conocida como *Serialización de copia simple*.

Un *método de replicación* debe ser capaz de manejar dos problemas: control de concurrencia y gerenciamiento de réplicas. Un *protocolo de control de concurrencia* asegura que no puedan ocurrir comportamientos incorrectos como resultado del acceso concurrente a los datos a partir de muchos clientes, y un *protocolo de gerenciamiento de réplicas* asegura que no puedan ocurrir comportamientos incorrectos a raíz de fallas en los sitios, particiones de la red o anomalías en la temporización de eventos. Muchos métodos de replicación tratan estos dos problemas con mecanismos independientes: en el bajo nivel, el acceso a las componentes individuales es sincronizado por un protocolo de control de concurrencia estándar, y en el alto nivel el estado de un objeto distribuído es reconstruído por un protocolo de gerenciamiento de réplicas a partir de sus componentes distribuídas, sin importar la concurrencia.

Una técnica para preservar la consistencia en presencia de fallas y concurrencia es organizar las operaciones en procesos secuenciales llamados *transacciones*. Estas son atómicas, esto es, serializables y recuperables. Por otro lado, se espera que las transacciones ejecuten operaciones confiables y consistentes [OVA91]. Estos dos atributos de las transacciones se obtienen gracias a las siguientes cuatro propiedades, conocidas como ACID:

- Atomicidad: se refiere al hecho de que una transacción sea tratada como una única operación. En consecuencia, todas las acciones de una transacción terminan o no termina ninguna de ellas.
- Consistencia: hace referencia a la correctitud de la transacción. En otras palabras, una transacción transforma la base de datos de un estado consistente a otro también consistente.

- Aislamiento: esta propiedad requiere que cada transacción vea a la base de datos en forma consistente todo el tiempo. En otras palabras, la ejecución de una transacción no puede revelar sus resultados a otras transacciones concurrentes hasta que no haya finalizado.
- Durabilidad: es la propiedad de las transacciones que asegura que una vez que la transacción finaliza exitosamente, sus resultados son permanentes y no pueden ser borrados de la base de datos.

La *serialización* significa que la ejecución de una transacción nunca se solapa o interfiere con la ejecución de otra, y la *recuperación* significa que una transacción termina completamente y en forma exitosa, o no tiene efecto. Si el efecto de una transacción se transforma en permanente, debido a su éxito, se dice que está *commit*.

Replicación y confiabilidad. Ventajas y desventajas

En relación con solucionar muchos de los problemas inherentes a las comunicaciones en BDD, los datos son copiados y almacenados en múltiples bases de datos, usualmente en plataformas de hardware diferentes. La principal razón para replicar información tiene que ver con la confiabilidad y con maximizar la velocidad de acceso. En un entorno cliente servidor es difícil que todos los usuarios dispongan de toda la información en el sitio en que se origina el requerimiento. También es difícil balancear los requerimientos de procesamiento entre usuarios de datos "livianos" (sistemas de procesamiento de transacciones on-line) y usuarios "pesados" (sistemas para soporte en la toma de decisiones), típicamente denominados Data Warehouses, aunque estos en general utilizan otra base de datos para su funcionamiento. En estos casos la replicación sería una buena elección. Por otro lado, ante una falla en parte del sistema, con la información replicada, esta permanecería disponible al resto de los usuarios en otras localidades.

En oposición a los beneficios de la replicación se encuentra la tarea de mantener las réplicas mutuamente consistentes: Un protocolo para control de réplica es esencialmente un protocolo para "sincronizar" las operaciones de lectura y escritura sobre los objetos replicados a través de diferentes transacciones concurrentes. Para asegurar la serialización, una lectura y una escritura de dos copias diferentes de los datos (residentes en diferentes nodos del sistema) deben poder ejecutarse en forma concurrente, obteniendo los mismos resultados. Hay que tener en cuenta que dos

operaciones de escritura en dos copias diferentes de los datos no deben ser permitidas en forma concurrente. [SAH96]

Alternativas de Replicación

Asumiendo que una base de datos está fragmentada en forma apropiada, hay que decidir como distribuir los fragmentos en los sitios de la red. Cuando los datos son alocados pueden ser replicados (múltiples copias) o mantenidos con copia simple. Las razones para replicar son básicamente dos: confiabilidad y eficiencia.

Una base de datos distribuída no replicada, usualmente llamada *base de datos particionada*, contiene fragmentos que son alocados en los sitios de la red, y existe una sola copia de cada fragmento. En el caso de replicación, la base de datos puede aparecer completamente en cada sitio, denominándose *base de datos totalmente replicada*, o solamente algunos fragmentos pueden estarlo, obteniendo una *base de datos parcialmente replicada*.

A continuación se muestra una tabla en la que se comparan las diferentes alternativas de replicación con respecto a las implementaciones de las funciones de los DBMS, estableciendo el grado de dificultad para mantener ciertas características deseables de las bases de datos [OVA91]. Hay que tener en cuenta que esta tabla no intenta indicar la performance de los diferentes grados de replicación, sino que indica comparativamente la viabilidad y facilidad de implementación de los diferentes aspectos de un DBMS.

	Totalmente replicada	**Parcialmente replicada**	**Particionada**
Procesamiento de consultas y actualizaciones	Fácil	Misma dificultad ↔	
Gerenciamiento de directorios	Fácil o inexistente	Misma dificultad ↔	
Control de Concurrencia	Moderado	Difícil	Fácil
Confiabilidad	Muy alta	Alta	Baja
Viabilidad de implementación	Aplicable	Real	Aplicable

Caracterización de los protocolos de replicación

Según [PSM98], un esquema de replicación se caracteriza por cuatro parámetros básicos:

1. Propiedad
2. Propagación
3. Refresco
4. Configuración

El parámetro *propiedad* (según [GHOS96]) define el permiso para actualizar una copia replicada. Si la copia es actualizable es llamada copia primaria, de otra forma es llamada copia secundaria.

El parámetro *propagación* define cuando la actualización a una réplica debe ser enviada a los nodos que almacenan otras copias del mismo objeto.

El parámetro *refresco* define el scheduling de las transacciones de refresco. Si una transacción de refresco es ejecutada tan pronto como es recibida se dice que la estrategia es inmediata. El par formado por los parámetros *refresco* y *propagación* define la estrategia de propagación de las actualizaciones.

El parámetro *configuración* caracteriza los nodos y la red.

A continuación se detallan los métodos de propagación de actualizaciones, los métodos de regulación de actualizaciones, y finalmente los protocolos de replicación.

2.1.- Métodos de Propagación de Actualizaciones

Para clasificar los Métodos de Propagación de Actualizaciones, se utiliza un parámetro basado en cuando tiene lugar la propagación de la actualización. La propagación de las actualizaciones puede ser realizada dentro o fuera de los límites de la transacción original. En el primer caso se denominan esquemas *Eager* o *Sincrónicos*, mientras que en el segundo son los llamados esquemas *Lazy* o *Asincrónicos*.

Esquemas *Eager* o *Sincrónicos*

Los esquemas de propagación *Eager* mantienen las réplicas sincronizadas en todos los nodos en todo momento, actualizando todas las réplicas como parte de una transacción atómica al momento que se actualiza la réplica originalmente requerida (Fig. 2.1).

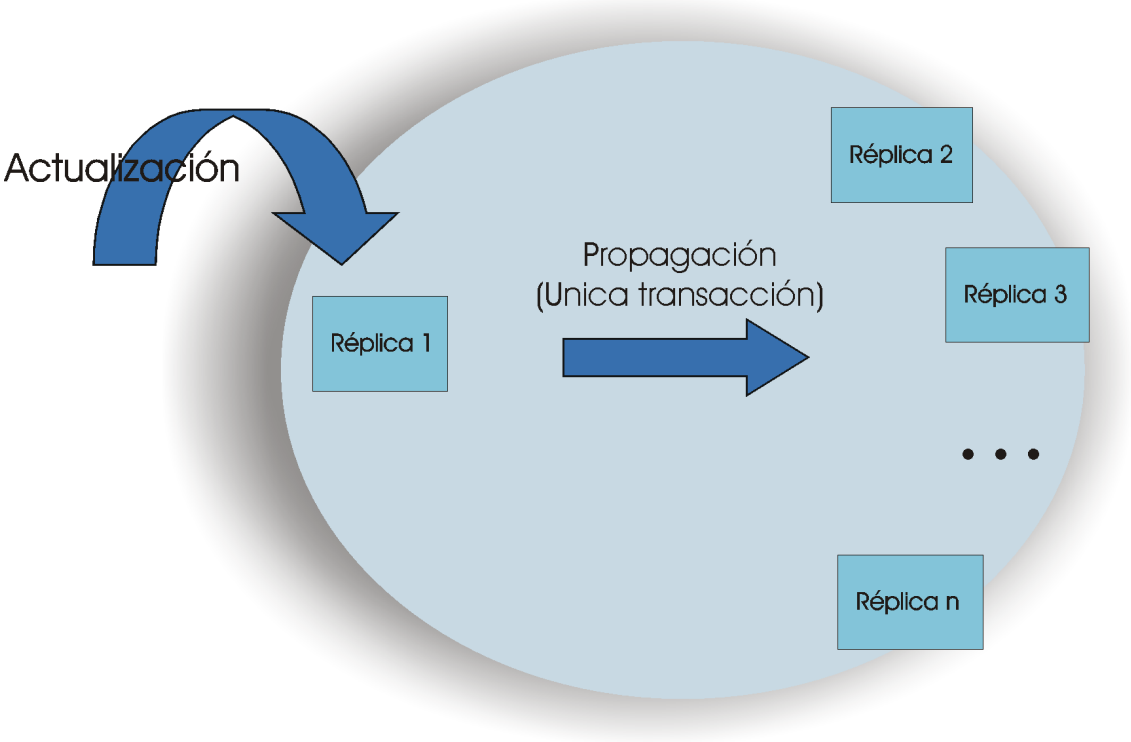


Fig. 2.1: Esquema *Eager*

Ventajas del esquema *Eager*

1. No hay inconsistencias entre los datos replicados. Al realizarse la actualización de todos los objetos replicados como parte de una sola transacción, es imposible que dos réplicas del mismo objeto sean diferentes.
2. La replicación *Eager* usa en general un esquema de lockeo para detectar y regular la ejecución concurrente. Por otro lado, se retrasan o abortan transacciones no finalizadas si su finalización afecta o viola la serialización. Los

lockeos detectan potenciales anomalías, y las convierten en Waits o Deadlocks. De esta forma se garantiza la consistencia de los datos y la ejecución concurrente en todo momento.

3. No hay necesidad de reconciliaciones. Las reconciliaciones son mecanismos para resolver conflictos cuando la ejecución concurrente de dos o más operaciones de actualización afectan la serialización y la consistencia de los datos. En un esquema *Eager*, al tener la actualización como parte de una transacción atómica, no habrá posibilidades de inconsistencia.

La configuración de la red y el esquema de distribución de los datos es importante a la hora de definir el esquema de replicación. La realidad indica que algunos nodos de la red podrían estar desconectados la mayor parte del tiempo, mientras que otros no. Por la característica transaccional de una actualización, las lecturas a nodos conectados generan datos actualizados, mientras que las lecturas a nodos desconectados podrían obtener datos desactualizados. Esto último puede prevenirse de dos formas:

1. Prohibir actualizaciones si existe algún nodo desconectado.
2. Intentar algún esquema de voto por mayoría para prevenir actualizaciones cuando existen nodos desconectados.

Aunque todos los nodos estén conectados todo el tiempo, las actualizaciones pueden fallar debido a deadlocks que previenen errores en la serialización.

Desventajas de la replicación *Eager*

1. Los nodos móviles no pueden usar el esquema cuando están desconectados. Los sistemas con replicación Eager sencillos prohíben las actualizaciones si al menos un nodo está desconectado.
2. La probabilidad de deadlock y consecuentemente transacciones fallidas, se incrementa rápidamente con el tamaño de las transacciones y el número de nodos. La justificación de este punto se detalla en [GHOS96].
3. El número de operaciones en las transacciones aumenta con el grado de replicación.
4. Se reduce la performance en actualizaciones y se incrementan los tiempos de respuesta de las transacciones, porque se agregan transacciones y mensajes a la transacción original.

Con un esquema puro, no hay solución aparente a estos problemas. Los esquemas *Eager* son difíciles de escalar mas allá de un número pequeño de sitios.

Esquemas *Lazy* o *Asincrónicos*

Los esquemas de propagación *Lazy* realizan la actualización original en una transacción, y la propagación de la actualización a las demás réplicas se realiza en forma asincrónica y en transacciones diferentes, una vez que la transacción original finaliza (Fig. 2.2).

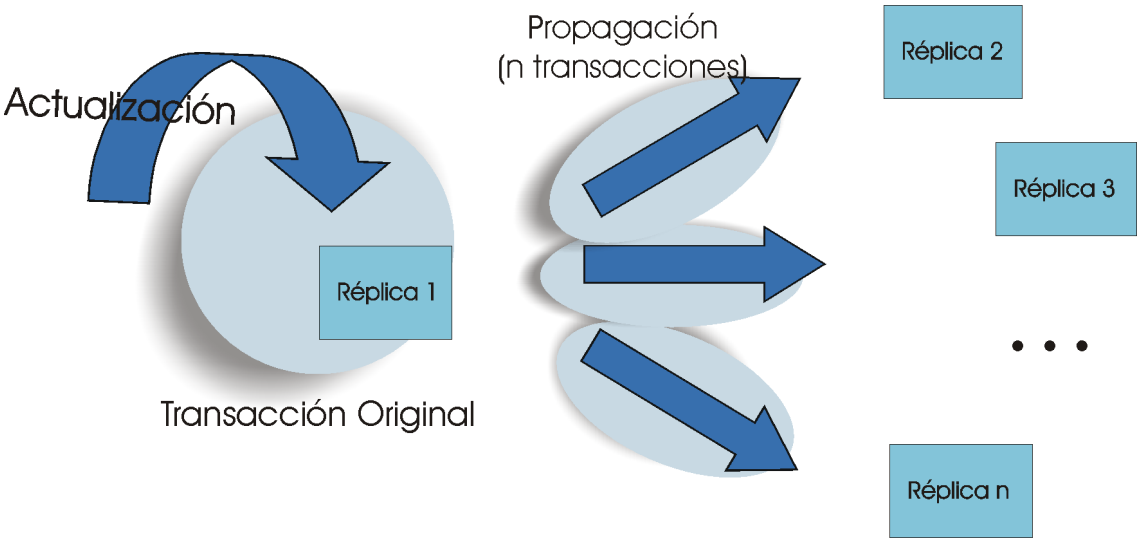


Fig. 2.2: Esquema *Lazy*

Ventajas de los esquemas *Lazy*

1. En contraposición con los esquemas de replicación *Eager*, los esquemas *Lazy* son una buena alternativa para aplicaciones móviles.
2. Algunos sistemas permanentemente conectados utilizan replicación *Lazy* para mejorar los tiempos de respuesta.
3. El tamaño efectivo de operaciones por transacción se reduce y la performance total del sistema mejora, produciendo menos deadlocks.

Desventajas de los esquemas *Lazy*

1. Un problema con la replicación *Lazy* en la mayoría de los sistemas comerciales es que estos pueden conducir fácilmente a ejecuciones no serializables. Esto sucede si un mismo dato es actualizado concurrentemente en dos sitios diferentes, resultando así un conflicto en la actualización. A veces se utiliza un esquema de control de concurrencia multi-versión para detectar comportamientos no serializables. Pero aun así, si se detecta un problema en la

serialización, algunas réplicas ya podrían haber sido actualizadas. No hay forma automática de revertir las actualizaciones realizadas. Una persona o programa debe *reconciliar* las transacciones conflictivas. En los sistemas comerciales son denominadas *reglas de reconciliación*.

Análisis comparativo de esquemas *Eager* y esquemas *Master*

Cuando hay datos replicados, una transacción simple en un nodo puede aplicar sus actualizaciones remotamente (a las demás réplicas del dato actualizado) ya sea como parte de una única transacción (*Eager*) o en transacciones separadas (*Lazy*). En ambos casos, si los datos están replicados en N nodos, la transacción realiza N veces más trabajo [GHOS96]. La diferencia entre las transacciones realizadas de acuerdo al esquema elegido se muestra la siguiente figura (Fig 2.3):

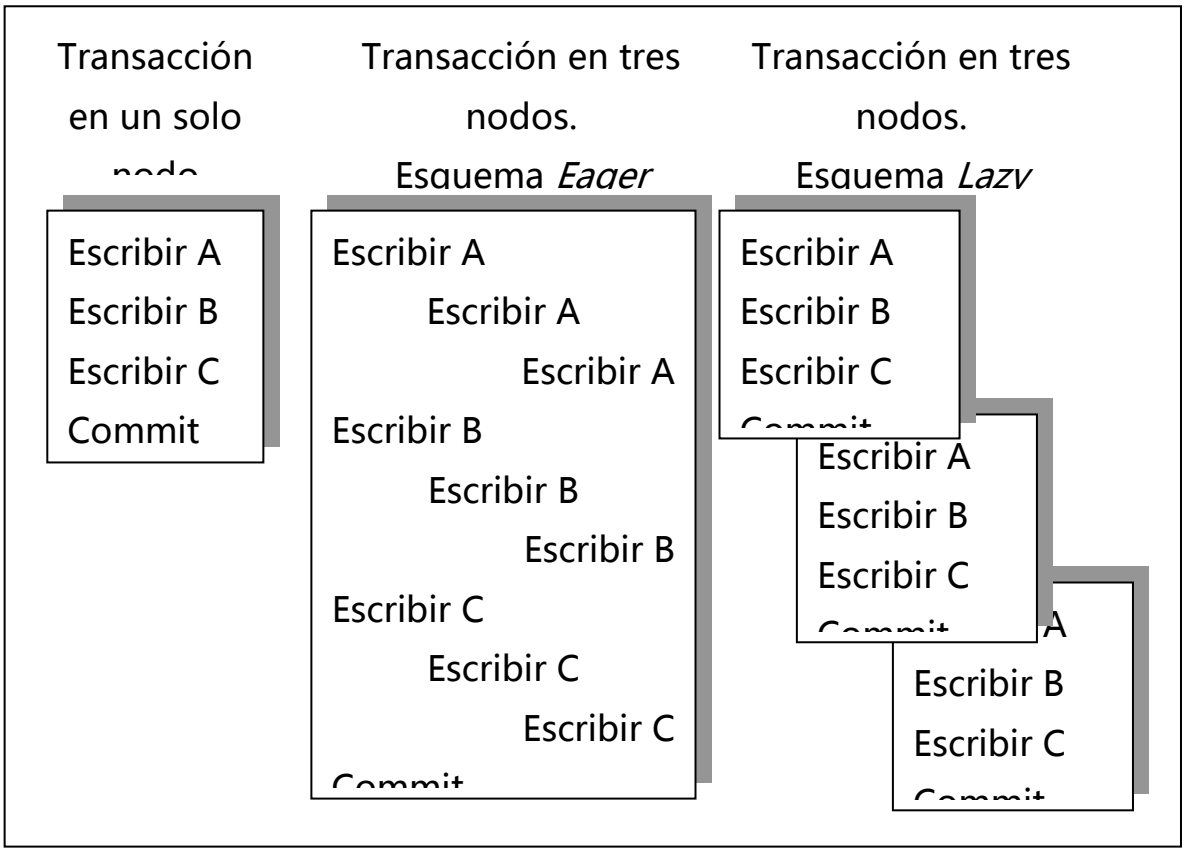


Fig. 2.3: Esquema comparativo evaluando cantidad de transacciones

2.2.- Métodos de Regulación de Actualizaciones

Los métodos de regulación de actualizaciones se basan en la propiedad de los datos. A los esquemas donde existen nodos propietarios de los datos se los denomina *Master – Slave*, mientras que si no existen propietarios se los denomina esquemas *Group*.

Esquemas Master – Slave

Los protocolos más simples se basan en la propiedad de los datos. Los algoritmos de copia primaria designan una copia del objeto de datos como la *copia primaria*. El nodo que mantiene dicha copia es responsable de proveer acceso mutuamente exclusivo. Cualquier nodo que mantiene una copia no primaria, denominada *copia esclava*, que desea ejecutar una actualización debe pedir permiso al dueño de los datos su consentimiento para hacerlo. Una vez que la copia primaria es actualizada, la actualización es propagada a todos los nodos que mantienen copias esclavas.

En un esquema *Master – Slave* cada objeto posee un nodo propietario, denominado *master*. Solamente el nodo master puede actualizar la copia primaria del objeto. Todas las otras réplicas son de sólo lectura, y si los nodos desean actualizar el objeto del cual no son propietarios deben recurrir al nodo *master* para que realice la actualización. Luego esa actualización es propagada en una segunda etapa a los nodos *slave* (Fig. 2.4).

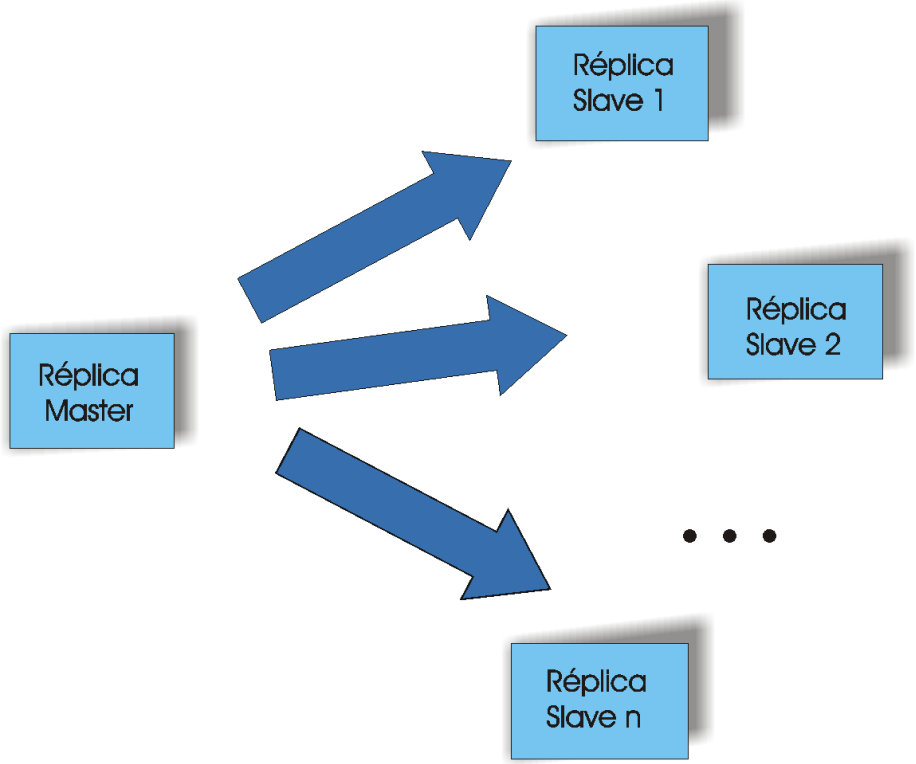


Fig. 2.4: Esquema *Master – Slave*

Una posible implementación para un esquema *Master – Slave* es tener una base de datos master, la cuál es usada para actualizaciones, y muchas bases de datos de consulta que son refrescadas desde la master en forma periódica, por ejemplo, una vez al día. El método *Master – Slave* usualmente requiere de una base de datos de cambios, la cual registra todos los cambios a la base de datos maestra. En el momento de propagar los cambios a las réplicas se toman los datos de la misma. Cuando todas las réplicas son actualizadas, se eliminan todas las entradas de la base de datos de cambio. [BUR94]

Ventajas de los esquemas *Master – Slave*

La principal ventaja de los esquemas *Master – Slave* es la simplicidad. Todos los sistemas que utilizan este esquema para la regulación de las actualizaciones son fáciles de implementar. Además, teniendo las actualizaciones concentradas en un solo nodo de la red distribuída se simplifica la tarea de control de concurrencia.

Ahora bien, esta ventaja trae consigo una serie de desventajas, que se detallan a continuación.

Desventajas de los esquemas *Master – Slave*

1. No todas las réplicas tienen la misma funcionalidad en iguales condiciones: las copias slave solo pueden realizar operaciones de lectura. Cada vez que en el nodo existe la necesidad de actualización debe sincronizarse con la copia master para realizarla.
2. La implementación de la copia master y las slaves puede ser diferente. En consecuencia, los dialectos utilizados por cada uno también. Esto debe resolverse al momento de establecer la forma de comunicación entre los nodos.
3. Otro problema tiene que ver con el timing: cuanto tiempo no están sincronizadas la copia master y las copias slave. Puede resolverse teniendo en cuenta diferentes alternativas:
    - Se actualizan las copias slave periódicamente (una vez por día, una vez por hora, etc.). La decisión de elegir la periodicidad de actualización depende de las reglas de negocios, y de la necesidad de tener datos actualizados en forma permanente.
    - Cada copia slave avisa a la master cuando está libre para ser actualizada.
    - Esperar a que todas las copias slave estén libres para proceder a la actualización en forma masiva.

4. Si una de las copias slave se torna no disponible, ¿se procede con los cambios? Hay dos alternativas posibles: actualizar solamente las copias disponibles, o esperar a que todas lo estén. Esto puede llevar a inconsistencias en la base de datos, debido a que los nodos con copias secundarias podrían estar desactualizados y atendiendo consultas.
5. Es difícil mantener la integridad referencial.
6. La disponibilidad puede verse afectada de dos formas:
   - Al haber un único nodo master de los datos, en él se concentran todos los requerimientos de actualizaciones. Esto puede transformarse en un cuello de botella que debe resolverse al momento de implementar.
   - Si un nodo master no puede ser accedido, ya sea por una falla del nodo o una falla en un link hacia el mismo, las copias master no pueden aceptar actualizaciones hasta su recuperación. De esta forma se introduce un único punto de fallas. La recuperación puede consistir en una simple espera de su disponibilidad o la búsqueda de un nuevo propietario del dato. Esto afecta directamente la disponibilidad y hace al esquema difícil de usar en aplicaciones móviles [GHOS96].

Todas estas desventajas enunciadas pueden de alguna forma ser tenidas en cuenta al momento de implementar un esquema *Master – Slave*, a fin de considerarlas e implementar soluciones a las mismas.

En cuanto a la escalabilidad, a medida que aumentan las réplicas, aumenta la demora en la propagación de las actualizaciones y a veces esto crea cargas desequilibradas entre las réplicas. Los esquemas *Master – Slave* experimentan O(N) conflictos de actualización, donde N es el número de réplicas [GHOS96]. Una solución es conectar las réplicas en una estructura de árbol, localizando el master en la raíz, y las actualizaciones se van propagando hacia abajo desde la raíz. Esto acorta el retraso de propagación de O(N) a O(log N), con N igual al número de réplicas, y reduce la carga sobre el master a un nivel constante.

Variaciones del esquema *Master – Slave*

Existen diferentes métodos basados en el esquema *Master – Slave*, pero de alguna manera difieren en la manipulación de los datos. Entre ellos se destacan:

- **Sitio Primario o sin fragmentación del sitio primario**: también conocido como *Master – Slave* con primario no fragmentado, debido a que todas las actualizaciones se realizan en un solo sitio (el primario o master) y los cambios luego se propagan a las réplicas slave.
- **Sitio con respaldo**: se designa un segundo sitio como sitio de respaldo, a fin de ser más tolerante a fallas.
- **Copia primaria o con distribución de los fragmentos primarios**: con este método se intenta distribuir la carga en varios sitios, es decir que las copias master de cada ítem de datos puede almacenarse en diferentes sitios. Cada uno actúa como master para un conjunto particular de datos.
- **Con migración del propietario**: consiste en migrar el master del dato sobre diferentes réplicas a partir de un determinado criterio, basado generalmente en el factor tiempo.

Las ventajas y desventajas de cada una de estas variaciones se describen en detalle en [Len01].

Esquemas Group

Los esquemas *Group*, también denominados Update-Everywhere, permiten que cualquier nodo pueda realizar actualizaciones sobre sus réplicas locales, logrando tanto acceso de lectura como de escritura sobre los mismos [GHOS96]. Cuando una copia es actualizada, el nuevo valor debe ser propagado a las demás réplicas (Fig. 2.5).

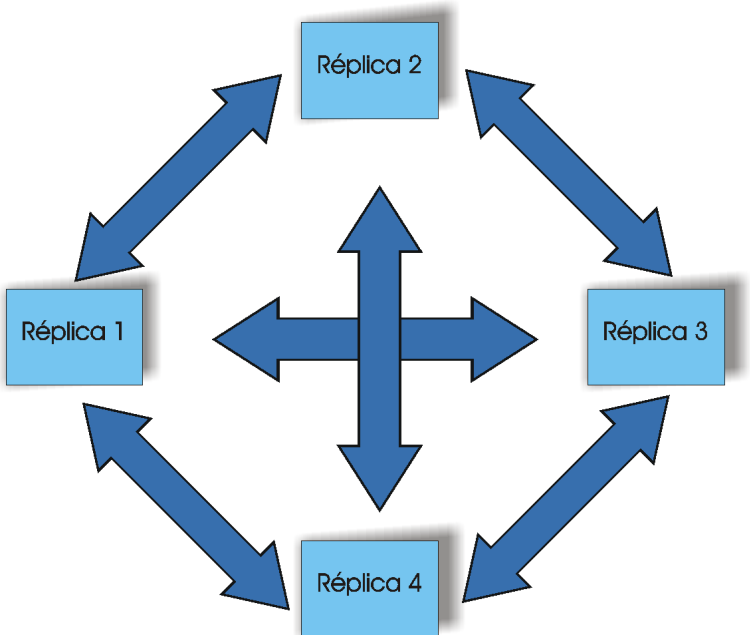


Fig. 2.5: Esquema Group

Ventajas del esquema *Group*

1. Soluciona los problemas de propiedad de los datos inherente al esquema *Master – Slave*. Se puede hablar de igualdad de réplicas, sin ningún derecho, propiedad o tratamiento especial.
2. Mayor disponibilidad de los datos, ya que cualquier réplica se puede sincronizar con cualquier otra, y toda actualización es aplicable a cualquier réplica accesible. Ademas se eliminan los cuellos de botella ya que las actualizaciones se realizan en el sitio que las origina, y luego se propagan a las demás.
3. Aumenta la funcionalidad de los nodos, ya que se pueden comunicar cualquier par de réplicas, haciendo al modelo más poderoso.
4. Se torna más flexible la etapa de diseño, ya que no hay necesidad de preocuparse por donde y qué operaciones pueden realizar los usuarios, ya que no hay concepto de nodo central o master y nodos remotos o slaves.

Desventajas del esquema *Group*

1. En estos esquemas pueden ocurrir actualizaciones concurrentes diferentes de dos copias del mismo dato, cosa que no sucede en los esquemas *Master – Slave*, pudiendo llevar esto a un conflicto de actualizaciones. En [GHOS96] se sostiene que los algoritmos de replicación basados en este esquema no pueden soportar muchas réplicas porque se experimenta $O(N^2)$ de conflictos de actualización, aumentando notablemente estos a medida que aumenta el grado de replicación. Esto significa que los sistemas con este esquema de replicación no son fácilmente escalables.
2. Si no se es cuidadoso con el diseño, se puede afectar la performance, obteniendo tiempos de respuesta peores que en los esquemas de copia primaria. Hay que tratar de balancear la carga de trabajo de todos los sitios, para que no se produzcan sobrecargas en ninguno de ellos.

Los esquemas de replicación basados en la propiedad dinámica de los datos han introducido en algunos casos el concepto de VOTO. El algoritmo *Majority Consensus Voting* (Robert Thomas) [ZHHO], está basado en la restricción de que para ejecutar una actualización el nodo demandante debe obtener el permiso de la mayoría de todos los nodos del sistema distribuído. El hecho de no haber control centralizado sobre los datos hace al algoritmo más robusto comparado con el esquema de copia primaria. Si bien se

soluciona el problema de propiedad de los datos, aumenta el costo de comunicación, y si hay problemas en las comunicaciones que particionan la red, y ninguna de las particiones contiene la mayoría de los nodos no podrían realizarse las actualizaciones.
Para solucionar el problema de las particiones sin mayoría de nodos, [SPL92] propone una variante al algoritmo del *voto por mayoría* denominado *Voto con Testigos*.
Voto con testigos reduce el número de réplicas necesarias para alcanzar un nivel de tolerancia a fallas reemplazando algunas réplicas por los testigos que contienen el estado del objeto replicado. Se reduce significativamente los requerimientos de almacenamiento, ya que dos réplicas y un testigo proveen la misma disponibilidad que tres réplicas completas utilizando el 33% menos de espacio de almacenamiento. Además propone una alternativa "económica" para la implementación de los testigos, que los denomina *testigos volátiles*.

2.3.- Protocolos de Replicación Clásicos

La combinación entre esquemas de replicación, en particular cruzando los que se basan en los parámetros de propagación y propiedad, da como resultado la siguiente tabla:

Propagación vs. Propiedad	Lazy	Eager
Group	N Transacciones N objetos propietarios	1 transacción N objetos propietarios
Master	N Transacciones 1 objeto propietario	1 transacción 1 objeto propietario

Protocolo Lazy Master – Slave

Como cualquier esquema *Master – Slave*, existe un nodo o sitio master para cada item de datos. Las actualizaciones son primero realizadas en el master en una única transacción (esquema *Lazy*) y luego propagadas a las demás réplicas, como parte de actualizaciones independientes. En cuanto a las lecturas, pueden realizarse en cualquier sitio, sin necesidad de recurrir al master del dato, con la restricción que en el sitio master siempre se accederá al valor correcto del dato, mientras que sobre las copias slave podrían leerse valores incorrectos, que aun no han sido actualizados por la propagación de la transacción original.

En [GHOS96] se propone por simplicidad, asumir que el nodo que origina la transacción dispare la actualización a las réplicas slave luego que la transacción master ha finalizado. Las actualizaciones en los slaves son etiquetados con un *timestamp*, de manera de asegurar que todas las réplicas convergen a un mismo estado final.

Los sistemas *Lazy Master - Slave* no poseen reconciliación ante fallas. Los conflictos se resuelven con esperas waits y deadlocks.

Aplicaciones

La replicación *Lazy Master - Slave* no es apropiada para aplicaciones móviles, ya para que un nodo que puede actualizar un objeto debe estar conectado el nodo propietario del objeto, y participar de una transacción atómica con el mismo.

Sin embargo , es el esquema de replicación más usado en aplicaciones de bases de datos comerciales.

Protocolo *Lazy Group*

Como cualquier esquema *Lazy*, cuando la transacción original finaliza, se envía la actualización a cada una de las otras réplicas. Cualquier réplica que posea los datos puede actualizarlos o leerlos (característica de los esquemas *Group*), y en transacciones diferentes propagarlas a las demás réplicas.

La propagación asincrónica de las actualizaciones debido a la característica de los esquemas *Lazy* hace que exista la posibilidad que en algunos sitios se lean valores viejos, que aún no han sido actualizados.

Aparece en este protocolo un nuevo problema, los conflictos de actualizaciones. Esto ocurre cuando dos o más transacciones que operan sobre distintas réplicas de un dato finalizan casi simultáneamente, generando un conflicto de actualización sobre dicho item. Por ejemplo, si en en el sitio S_1 se actualiza el dato *d*, y además en el sitio S_2 se actualiza otra réplica del dato *d*, y se asume que ambas transacciones finalizan casi simultáneamente, al momento de propagar dichas actualizaciones se detectará el conflicto entre ambos datos. El protocolo de replicación debe detectar esto y reconciliar las dos transacciones de manera de no perder ninguna de las actualizaciones.

Estos conflictos ocurren debido a que cuando se usa este modelo no hay aislamiento de las transacciones desde una perspectiva global, es decir, la transacción se ejecuta como si no hubiese ninguna otra transacción concurrente.

Una reconciliación es el mecanismo mediante el cuál se resuelven los conflictos de actualizaciones. Estos mecanismos pueden ser manuales, delegando la tarea a un administrador o al mismo usuario, o mediante transacciones de compensación generadas por el sistema, que deshagan las transacciones finalizadas que necesitan reconciliación para mantener la consistencia de los datos sobre todas las réplicas.

El uso de transacciones compensatorias hace que a veces las actualizaciones no sean durables. En consecuencia, la pérdida de la durabilidad crea un efecto en el cual un usuario puede tomar una decisión sobre información que es subsecuentemente la perdedora en la resolución de un conflicto. Dependiendo del grado de consistencia deseado puede suceder que las transacciones secundarias también requieran acciones compensatorias para asegurar la consistencia de la base de datos, y así sucesivamente. [BUR97].

En [GHOS96] se propone que para detectar y reconciliar actualizaciones dentro de un protocolo *Lazy – Group* transaccional se utilicen *timestamps*. Cada objeto lleva el *timestamp* de su más reciente actualización. Además cada actualización lleva el nuevo valor y este es adosado al viejo *timestamp* del objeto. Cada nodo detecta las actualizaciones de una réplica que pueden sobrescribir actualizaciones anteriores ya finalizadas. El nodo testea si el *timestamp* de la réplica local y el viejo *timestamp* de la réplica son iguales, en ese caso la actualización es segura: el *timestamp* de la réplica local avanza al *timestamp* de la nueva transacción y el valor del objeto es actualizado. Si los *timestamps* no son iguales, la transacción puede ser "peligrosa". En tal caso el nodo rechaza la transacción entrante y la propone para reconciliación.

En el caso de las bases de datos comerciales, en general se han desarrollado reglas de reconciliación para la resolución de conflictos, como Oracle 7, que provee una gama de 12 reglas de reconciliación para resolver actualizaciones conflictivas. Además los usuarios pueden programar sus propias reglas, inclusive con prioridades, distintos en cada sitio.

Aplicaciones

Es muy utilizado en aplicaciones móviles, donde el procesamiento debe realizarse usando componentes generalmente desconectadas. La idea básica es permitir a los usuarios leer o actualizar los datos mientras están desconectados, reconciliando las modificaciones con las demás réplicas cuando se reconectan al sistema, por ejemplo, Lotus Notes, que utiliza el protocolo *Lazy Group*.

Este esquema es aplicable a sistemas en que es aceptable cierto grado de inconsistencias. También es útil para aplicaciones donde las actualizaciones son operaciones de inserción o altas.

Protocolo *Eager Master - Slave*

Como en los esquemas *Master – Slave*, una operación de actualización siempre se comienza en la copia master del dato, y una vez que la actualización del mismo finaliza, se propaga dicha actualización a todas las copias slaves. En este caso, al tratarse además de un esquema *Eager*, la actualización de la copia master y de todas las copias slaves se realizan en una única transacción, de manera atómica. La copia master debe esperar la confirmación de todas las copias slaves. En cuanto a las lecturas, estas pueden ser realizadas desde cualquier sitio, y siempre se obtendrá la última versión del dato leído.

Los protocolos *Eager Master – Slave* pueden incluir la ejecución de un protocolo para verificación de consistencia, que puede ser el Two-Phase Commitment Protocol (2PC). Con este esquema se puede garantizar que si el sitio primario falla, todas las transacciones activas serán abortadas.
Si bien la replicación *Eager* tiende a provocar deadlocks, en [GHOS96] se propone tener un master para cada objeto, de manera de ayudar a la replicación Eager a evitarlo, reduciéndolos tanto como en un sistema de un solo sitio.

Aplicación
Ingres distribuído fue una de las primeras soluciones que usaron el modelo *Eager Master – Slave*, utilizando el método de copia primaria.
Actualmente, estos protocolos de replicación son usados solamente en tolerancia a fallas para implementar soluciones Hot-Standby, donde bajo una operación normal los requerimientos de los usuarios son tratados por el sitio primario, enviando sus modificaciones de manera inmediata a la copia secundaria o backup. Debido a la propagación sincrónica, el backup es una réplica exacta del sitio primario, pudiendo tomar su lugar de inmediato ante cualquier falla.

Protocolo *Eager Group*

En estos protocolos, la regulación de las actualizaciones está basada en el esquema *Group*, donde tanto las actualizaciones como las lecturas pueden realizarse en forma local. Además, la propagación de las actualizaciones es sincrónica debido al esquema *Eager*, debiendo sincronizarse el sitio que origina la actualización con todos los sitios que poseen réplicas del dato actualizado. Este esquema tiene como ventaja que cualquier nodo que realiza lecturas obtiene datos actualizados, evitando la lectura de datos viejos.
Los conflictos en las actualizaciones se evitan mediante bloqueos o esperas. Estos protocolos evitan mediante esperas o bloqueos que dos actualizaciones generadas en diferentes sitios "choquen" al actualizar la misma réplica [GHOS96].
Los protocolos *Eager Group* son esquemas basados en *Serializabilidad de una Copia*, donde un objeto debe aparecer como una copia lógica y la ejecución concurrente es coordinada para que sea equivalente a una ejecución serial sobre una copia lógica, evitando de esta manera cualquier tipo de conflicto.
La implementación de los protocolos *Eager Group* puede basarse en Bloqueos Distribuídos o en Broadcast Atómicos.

En el caso de implementarlos a través de Bloqueos Distribuídos, los pasos en el proceso de actualizaciones son los siguientes:

1. Requerimiento del usuario en un sitio local. Comienza la transacción.
2. Coordinación de los sitios, utilizándose un Protocolo de Bloqueos, como el Two Phase Locked (2PL) [OVA91], mediante el cual se manda un requerimiento de bloqueo a todos los demás sitios que pueden otorgar o no el mismo. Hay que tener en cuenta que en esquemas con bloqueos distribuídos una réplica solamente puede ser accedida después que ha sido bloqueada en todos los sitios. Si el bloqueo es concedido por todos los sitios, se puede proceder a actualizar, sino, la transacción puede ser demorada y el requerimiento puede ser repetido más tarde.
3. Ejecución de la operación.
4. Coordinación del acuerdo, utilizando un protocolo de Commit Atómico, como el Two Phase Commit Protocol (2PC), para ejecutar o abortar la transacción en todos los sitios.
5. Respuesta al usuario, avisando que ha finalizado la transacción.

En cuanto a la implementación de los Protocolos *Eager Group* en base a Broadcast Atómicos, se realizan los siguientes pasos:

1. Requerimiento del usuario en un sitio local. Comienza la transacción.
2. Coordinación de los sitios: el sitio que origina la transacción distribuye (broadcast) el requerimiento a todos los demás sitios, con los cuales se coordina usando un orden dado por el Broadcast Atómico.
3. Ejecución de la operación.
4. Respuesta al usuario.

No es necesaria una coordinación del acuerdo, ya que el Broadcast Atómico brinda la propiedad de atomicidad.

Aplicación

Este protocolo de replicación ha sido ampliamente estudiado, en todos los casos logrando la serializabilidad de una copia, pero implementadas de manera diferente variando las características de performance [Len01], entre las que se encuentran protocolos conocidos basados en quorums, como ROWA (read only / write all), ROWAA (read one / write all available), o QC (Quorum Consensus).

En el marco de una gran cantidad de investigaciones se sostiene que estas aproximaciones no son muy relevantes en la práctica debido a sus problemas de

performance y escalabilidad [GHOS96]. Esto ha provocado una fuerte tendencia en los diseñadores a no utilizarlos, llevando a que estos esquemas no hayan sido prácticamente usados en productos comerciales.

2.4.- Protocolos de Replicación Híbridos

A modo de ejemplo, se detallarán dos protocolos de replicación híbridos, donde cada uno de ellos toma ventajas y mejora las desventajas de los protocolos de replicación básicos.

Según [GHOS96], un esquema de replicación ideal debería tener estas cuatro características:

1. **Disponibilidad y escalabilidad**: Proveer alta disponibilidad y escalabilidad a través de la replicación, evitando inestabilidad.
2. **Movilidad**: Permite nodos móviles para leer y actualizar la BD mientras están desconectados.
3. **Serialización**: Proveer ejecución de transacciones serializables de copia simple.
4. **Convergencia**: Proveer convergencia para evitar un "sistema engañoso" (system delusion)

La convergencia se define a través de la siguiente propiedad:

Propiedad de Convergencia: si no arriban nuevas transacciones, y si todos los nodos están interconectados, todos convergen al mismo estado de replicación luego de intercambiar las actualizaciones de las réplicas. El estado resultante contiene las actualizaciones finalizadas, y los más recientes reemplazos.

Los esquemas de replicación transaccionales más seguros (por evitar system delusion) son:

- Sistemas *Eager*
- Sistemas *Lazy Master-Slave*

Si bien no tienen problemas de reconciliación, tienen los siguientes inconvenientes:

1. Los objetos en la master no admiten actualizaciones si el nodo master no está accesible. Esto hace difícil la implementación en aplicaciones móviles.
2. Los sistemas *Master* son inestables si se incrementa la carga. El deadlock se alcanza rápidamente cuando hay mayor cantidad de nodos.
3. Los sistemas *Eager* y *Lazy Master – Slave* admiten serialización ACID.

Para evitar estos problemas es necesario cambiar la forma en que se usa el sistema.

Los sistemas de replicación *Lazy Group* son propensos a la reconciliación a medida que se agrandan. La reconciliación manual para resolución de conflictos es inviable. Una aproximación es deshacer todo el trabajo de cualquier transacción que necesita la

reconciliación. Esto hace a las transacciones atómicas, consistentes y aisladas, pero no durables. En todo sistema con estas características toda transacción es tentativa hasta que haya sido propagada a todas las réplicas. No es viable para aplicaciones móviles.

Protocolo de Replicación Híbrido *Two – Tier*:

Este protocolo está basado en solucionar los problemas referentes a las actualizaciones cuando uno o más nodos están desconectados. Es adecuado para sistemas móviles, donde algunos nodos pueden estar desconectados la mayor parte del tiempo.

La solución tiende a un esquema de replicación *Lazy Master - Slave* modificado:

- Cada objeto tiene un nodo propietario
- Los nodos móviles realizan actualizaciones tentativas.
- Cuando se conectan, estas pueden o no hacerse efectivas.

Esquema de Replicación *Two - Tier*

Básicamente, se manejan dos clases de nodos:

- **Nodos Móviles**: desconectados la mayor parte del tiempo. Almacenan una réplica de la BD y originan *transacciones tentativas*. Pueden ser propietarios de un grupo de items.
- **Nodos Base**: siempre están conectados. Almacenan una réplica de la BD. Son propietarios de la mayoría de los items.

Los items de datos en los nodos móviles se manejan en dos versiones:

- **Versión Master**: Es el valor más reciente recibido del objeto master. Si está desconectado esta versión puede ser vieja.
- **Versión tentativa**: La versión local del objeto puede ser modificada por transacciones tentativas. Es el valor local más reciente debido a actualizaciones locales.

Existen además dos clases de transacciones:

- **Transacción Base**: Trabaja solamente sobre datos master y producen nuevos datos master. Involucra como máximo un nodo móvil conectado y varios nodos base.
- **Transacción Tentativa**: Trabajan sobre datos locales tentativos. Producen nuevas versiones tentativas de los datos. Producen transacciones base que serán corridas más tarde en lo nodos base.

Para las transacciones tentativas se debe seguir una regla de alcance: deben involucrar objetos cuyos propietarios son los nodos base y objetos cuyos dueños son los nodos móviles que originan la transacción. Todos los nodos base y el nodo móvil deben estar en contacto cuando la transacción tentativa es procesada como una transacción base.
Una transacción base generada por una transacción tentativa puede fallar o puede producir resultados diferentes. Deberá existir algún criterio de aceptación que deberá chequear la transacción base cuando procesa la transacción tentativa. Depende esto de la aplicación y de las reglas de negocios.
En caso que la transacción base generada por una transacción tentativa falle, el nodo y la persona que originaron la transacción son informados de la falla y del porqué.

El esquema *Two-Tier*, durante una operación de un nodo conectado, opera igual a un sistema *Lazy – Master* con la restricción adicional que ninguna transacción puede actualizar los datos cuyos propietarios son nodos móviles. Esta restricción no es realmente necesaria si el nodo móvil está conectado.
Si se considera ahora el caso de un nodo desconectado, suponiendo que éste estuvo desconectado un lapso de tiempo considerable, tiene una copia de los datos obsoleta, y ha generado transacciones tentativas en esa base de datos local sobre los datos de los cuales es propietario.
Cuando un nodo móvil se conecta a un nodo base, el nodo móvil realiza las siguientes acciones:

1. Descarta las versiones tentativas de objetos ya que serán refrescadas en breve por los nodos master.
2. Envía las actualizaciones a las réplicas de objetos de los cuales es propietario.
3. Envía todas las transacciones tentativas a los nodos base para se ejecutadas en el orden en que fueron ocurriendo.
4. Acepta las actualizaciones de las réplicas de los nodos base.
5. Acepta el mensaje de éxito o falla de cada transacción tentativa.

Por otro lado, cuando un nodo base es contactado por un nodo móvil que acaba de conectarse, realiza las siguientes acciones:

1. Envía las transacciones de actualización de réplica demoradas.
2. Acepta las transacciones de actualización de los objetos de los cuales es propietario el nodo móvil.

3. Acepta la lista de transacciones tentativas, re-ejecuta cada una en el orden en que fueron ejecutadas en el nodo móvil. Durante esta re-ejecución se utiliza el esquema *lazy – master* tradicional. En función de los resultado obtenidos en la ejecución, envía al nodo móvil el aviso de éxito o falla de cada transacción.
4. Cuando el nodo base ejecuta con éxito una transacción tentativa del nodo móvil, esta es reenviada a cada nodo conectado, de acuerdo al esquema *lazy – master*.
5. Cuando todas las transacciones tentativas han sido reprocesadas como transacciones base, el estado del nodo móvil converge al estado del nodo base.

Las propiedades clave del esquema de replicación *Two-Tier* son las siguientes:

1. Los nodos móviles pueden ejecutar transacciones tentativas de actualización en la base de datos.
2. Las transacciones base ejecutan con serialización de copia simple de manera que el estado del sistema base master es el resultado de la ejecución serial.
3. Una transacción se transforma en durable cuando se completa una transacción base.
4. Las réplicas de todos los nodos conectados convergen al estado base del sistema.
5. Si todas las transacciones conmutan, no hay reconciliaciones.

Cuando se ejecuta una transacción base, el esquema de replicación *Two – Tier* se comporta como un esquema *Lazy – Master*. La frecuencia de las reconciliaciones será cero si todas ellas conmutan.

El procesamiento de las transacciones base puede producir resultados diferentes a los arrojados por la ejecución de las transacciones tentativas. Si bien hay casos de aplicaciones en que esto puede ser aceptable, en otros no lo es. El criterio de aceptación con referencia a este problema es específico de cada aplicación. El sistema de replicación solamente detecta la diferencia entre los resultados de la transacción tentativa y la base. Además, el sistema solamente ejecuta la transacción tentativa, y si esta se completa exitosamente y pasa el test de aceptación, entonces se asume que está todo bien y se propaga la actualización a las réplicas de la manera usual. Por otro lado, las transacciones deben ser diseñadas de manera que sean conmutativas.

Este esquema de replicación híbrido puede ser usado para obtener serilización pura si las transacciones base sólo leen y escriben objetos master.

El esquema de replicación *Two – Tier* soporta nodos móviles y combina los beneficios de un esquema de replicación *Lazy – Master* y un esquema de replicación local estilo *Eager* para los nodos desconectados.

Ampliando el cuadro presentado en la sección de protocolos de replicación básicos (sección 2.3 de este capítulo), se muestra la taxonomía de las estrategias de replicación en contraste con estrategias de propagación (*Eager* o *Lazy*) con la estrategia de propiedad (*Master* o *Group*):

Propagación vs. Propiedad	**Lazy**	**Eager**
Group	N Transacciones N objetos propietarios	1 transacción N objetos propietarios
Master	N Transacciones 1 objeto propietario	1 transacción 1 objeto propietario
Two-Tier	N+1 transacciones, 1 objeto propietario. Actualizaciones locales tentativas Actualizaciones basadas en Eager	

Protocolo Híbrido para control de réplicas

Este protocolo se describe en detalle en [ZHHO]. Básicamente propone combinar en el algoritmo de control de réplicas las ventajas de los esquemas *Master – Slave* y *Group* y mejorar las desventajas de ambos.

Se basa en la aproximación de *copia primaria* y en la aproximación de *voto por mayoría*. La idea es que durante una operatoria normal, el algoritmo responde a la aproximación de *copia primaria*. Ante un evento de falla en las comunicaciones que cause la partición de la red, se convierte en un híbrido entre copia primaria y voto por mayoría, minimizando el overhead de comunicación y manteniendo un alto nivel de disponibilidad.

Las dificultades que presenta este esquema son las siguientes:

1. Reelección del nodo primario: Si ocurre una partición en la red no se puede garantizar que exista solo un nodo primario en el sistema al mismo tiempo, más aun si la comunicación entre particiones no está disponible. Igualmente se puede prevenir que un nodo primario realice actualizaciones si no está en la partición de la mayoría, basándose en el algoritmo de voto por mayoría.

2. ¿Qué pasa con los pedidos de actualizaciones pendientes en el nodo primario y que hacemos cuando ocurre una partición en la red?

   Pueden ocurrir cuatro situaciones:

   2.1.- El nodo primario actual está en la partición mayoritaria, y el pedido pendiente está en la misma partición.

   2.2.- El nodo primario actual está en la partición mayoritaria, pero el pedido pendiente está en una partición diferente.

   2.3.- El nodo primario actual no está en la partición mayoritaria, pero el pedido pendiente está en la misma partición.

   2.4.- El nodo primario actual no está en la partición mayoritaria, pero el pedido pendiente está en la partición mayoritaria.

   Para la situación 1, el nodo primario finaliza el pedido pendiente, y luego notifica al que realizó el pedido que este ha sido completado. En la situación 3 se denega toda actualización y se notifica al nodo que realizó el pedido.

   Las situaciones 2 y 4 son más complicadas. La comunicación entre el nodo primario y la entidad que realizó el pedido no está disponible. En el caso de la situación 2, la actualización puede o no proceder basándose en que la respuesta a la entidad solicitante no podrá realizarse. En la situación 4 el pedido de actualización será denegado, pero la entidad solicitante no podrá ser denegada.

El diseño del algoritmo asume que si no está disponible la comunicación con el nodo primario, está en otra partición. Durante la operación normal del sistema, la comunicación con el nodo primario puede no existir directamente, pero si en forma indirecta a través de otro nodo. Por simplicidad, el algoritmo primero trata de llegar directamente al nodo primario. Si esta falla, no trata de comunicarse a través de otro nodo. También asume que todos los clientes y sus servidores padres estarán en la misma partición luego de una falla en la comunicación.

Capítulo 3: JAVA como herramienta de desarrollo

Introducción

Existen un número de características de Java las cuales permiten la creación de componentes de simulación estandarizados. La característica de Java más relevante es que es simple y es la propuesta más integral a la programación multithreaded. [OAW97]

Java abarca dos aspectos fundamentales:

- Una Plataforma
- Un lenguaje de programación.

Una *plataforma* es un ambiente de software o hardware sobre el que se ejecuta un programa. En particular, Java es sólo una plataforma de software que se ejecuta por encima de otras plataformas de software como por ejemplo los sistemas operativos. La función de la plataforma Java es aislar los programas Java del hardware. [JAV01]
La plataforma Java tiene dos componentes (Fig. 3.1):

- La Java Virtual Machine (JVM): es la base de la plataforma Java y puede ser incorporada a la mayoría de las plataformas de hardware. Posee el intérprete Java.
- La Java Application Programming Interface (Java API): es una colección de componentes que proveen una amplia gama de funcionalidades, como GUIs, I/O, etc. Está agrupada en paquetes o librerías de componentes relacionadas. La Java API se divide en dos grupos: la API básica y la API extendida.

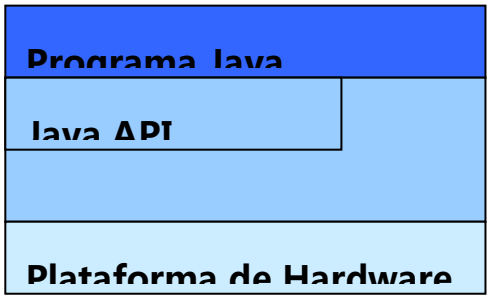


Fig. 3.1: Componentes de la plataforma Java

En cuanto al lenguaje de programación, Java posibilita el desarrollo de aplicaciones seguras, de alta performance, robustas sobre múltiples plataformas en redes heterogéneas y distribuídas. [JAV01]
Las principales características del lenguaje Java son:

- Orientado a objetos
- Distribuído
- Robusto
- Seguro
- Arquitectura neutral
- Multithreaded
- Alta performance
- Interpretado
- Dinámico

Si bien se puede desarrollar ampliamente cada una de estas características de Java como lenguaje de programación, se centrará la atención en los ítems que influenciaron en la elección de Java como lenguaje para desarrollo de este trabajo de grado.

Algunas características de Java

Orientación a Objetos en Java

Posee todas las características de un lenguaje orientado a objetos:

- Polimorfismo
- Encapsulamiento
- Binding Dinámico
- Herencia (solo herencia simple)

Además se enriquece la herencia simple de clases implementando interfaces. Una interfaz es una especificación de constantes y métodos sin implementación. Una clase puede implementar una o más interfaces, logrando así una similitud con la herencia múltiple.

Con esta característica, se pudieron definir en forma sencilla todos los componentes del proceso de generación y ejecución de trazas, los cuales se desarrollan y explican más adelante.

Multithread

Un *Thread* es un flujo de control secuencial dentro de un programa. Java provee múltiples threads en un programa, ejecutándose concurrentemente y llevando a cabo tareas distintas. A los threads también se los conoce como procesos livianos ó contextos de ejecución.

Multithread permite mejorar la interactividad y la performance del sistema.

La API Java contiene la clase *Thread* que soporta un conjunto de métodos para tratar threads: arrancar un thread (start), ejecutar un thread (run), interrumpir la ejecución de un thread (stop) y chequear el estado de un thread.

El soporte de Threads en Java incluye primitivas de sincronización. Si múltiples threads ejecutándose concurrentemente comparten recursos, necesitan mecanismos de sincronización para acceder al recurso. Esto se logra, declarando a los métodos synchronized.

Esta característica de multithread fue usada en este trabajo para la implementación de la simulación y posibilitar así la ejecución paralela de tareas específicas.

Alta Performance

El código Java es chequeado en compilación y en ejecución, en el primer caso por el Class Loader y en el segundo caso por el ByteCode Verifier. Por lo tanto el intérprete puede correr a toda velocidad sin necesidad hacer chequeos en ejecución.

Un garbage collection automático corre como un thread de baja prioridad, aprovechando los tiempos muertos del usuario, mejorando así la disponibilidad de memoria.

Esta característica influyó en la elección de Java para la implementación de este trabajo de grado, tanto por razones de tiempo inherentes a una simulación como por la cantidad de información que debe procesarse en este tipo de tareas. Si bien la programación de los algoritmos fue cuidadosa en este aspecto, el hecho que el lenguaje de base acompañe en esta tarea fue fundamental.

Threads

La plataforma JAVA soporta programas multhreading a través del lenguaje, de librerías y del sistema de ejecución.

Un thread es similar a un programa secuencial: tiene un comienzo, una secuencia de ejecución, un final y, en un instante de tiempo dado hay un único punto de ejecución. Sin embargo, un thread no es un programa. Un thread se ejecuta adentro de un programa.

Lo novedoso en threads, es el uso de múltiples threads adentro de un mismo programa, ejecutándose simultáneamente y realizando tareas diferentes.

Un thread es considerado un proceso “liviano” pues se ejecuta dentro del contexto de un programa y usa los recursos y el ambiente de ejecución del programa. Por ej. tienen su propia pila de ejecución y su contador de programa. Como un thread solamente se

ejecuta dentro de un contexto, a un thread también se lo llama contexto de ejecución. [JAV01]

La clase Thread forma parte del paquete java.lang y provee una implementación de threads independiente del sistema. Además provee el comportamiento genérico de los threads JAVA: arranque, ejecución, interrupción, asignación de prioridades, etc. El método run() implementa la funcionalidad del thread. El método run() de default, provisto por la clase Thread no realiza ninguna tarea específica, es allí donde el usuario programa las acciones relacionadas al comportamiento del thread que se está implementando.

Hay situaciones en las que los threads concurrentes comparten datos y en donde es necesario considerar el estado y coordinar las actividades de otros threads. Estos escenarios de programación se llaman productor / consumidor, y consisten en un productor que genera streams de datos que posteriormente serán consumidos por el consumidor.

Los threads que comparten recursos comunes deben sincronizarse de alguna manera. Las actividades del Productor y del Consumidor debe sincronizarse de dos formas:

Los threads no deben acceder simultáneamente al objeto compartido Bolsa. En Java se logra “bloqueando” el objeto compartido. Cuando un objeto fue “bloqueado” por un thread y otro thread invoca a un método sincronizado del objeto, éste thread se bloqueará hasta que el objeto sea “desbloqueado”.

Los threads deben coordinarse: el Productor debe indicarle al Consumidor de una manera sencilla que el valor está listo para consumirse y, el Consumidor debe tener alguna forma de indicarle al Productor que el valor ya fue recuperado. Para éste propósito la clase Thread provee los siguientes métodos: wait, notify y notifyAll.

Conclusiones

La importancia de la programación multithreaded para la simulación es que permite crear un thread que tiene control independiente de su comportamiento dentro de la simulación. [KHK98]

En ejecuciones multithreaded existen múltiples entidades inteligentes e independientes que comparten el control de la ejecución de instrucciones de programas, habiendo simultáneamente ejecuciones dentro del mismo programa.

La mayoría de los sistemas están descriptos como una colección de entidades independientes e inteligentes en vez de una simple entidad global que debe decidir cuándo y dónde debe alocar el control del programa.

Java tiene su soporte interno para la programación de redes distribuídas que provee la infraestructura restante necesaria para ejecutar los procesamientos de los experimentos de simulación distribuida sin hacer que la simulación dependa del sistema operativo o plataforma de hardware.

Capítulo 4: Diseño e Implementación del *SimReplica*

(Ambiente de Simulación para la toma de decisiones en el diseño de BDD)

4.1. Descripción general del problema

El simulador, denominado *SimReplica*, consta con varios estadíos, donde en cada uno de ellos se definen el modelo a evaluar y todos sus componentes. Estos son:

1. Modelo de datos del problema.
2. Esquema de distribución (nodos o localidades) y peso de cada conexión.
3. Esquema de replicación.
4. Generación de trazas de operaciones, en función de los porcentajes de altas, bajas, modificaciones y consultas a ser realizadas y evaluadas.
5. Ejecución de las trazas de operaciones definidas en la etapa anterior.
6. Visualización de los resultados obtenidos, para facilitar el análisis.

Para los estadíos 1, 2, 3 y 6 era necesario proveer un ambiente de trabajo amigable, con una interface adecuada, de manera que se realizó un diseño orientado a obtener una aplicación cliente/servidor. Esto fue finalmente implementado en Visual Basic 6.0, con el soporte de una base de datos Access.

Los estadíos 4 y 5 requerían la implementación de múltiples hilos de ejecución (threads), con una administración de la memoria adecuada y eficiente, de manera que se decidió la utilización de Java para esta etapa.

Finalmente, el ensamble adecuado de las dos implementaciones (VB y Java) dio por resultado la herramienta de software que se adjunta con este trabajo de grado.

<u>La simulación</u>

Una simulación comienza con el ingreso del modelo de datos que se desea evaluar. Esto implica:

1. Ingreso de las características generales del modelo.
2. Ingreso de las tablas de datos que conforman el modelo.
3. Por cada tabla, ingreso de los campos que la componen, con su tipo de datos y tamaño (en caso que sea requerido).
4. Ingreso del porcentaje de operaciones que aceptará el modelo, es decir, sobre un 100% de operaciones, el porcentaje de altas, bajas, modificaciones y consultas. A su vez, será necesario definir estos porcentajes por tablas, ya que en un modelo real las

actualizaciones seguramente afectarán en mayor proporción a algunas tablas, y no en forma equitativa o igualitaria a todas.

5. Definición de la red de nodos en los cuales se basará la distribución de las tablas. No es posible que un nodo esté físicamente desconectado. Este tipo de particionamiento de red haría imposible la utilización del nodo y, por ende, totalmente innecesaria su definición como integrante del modelo.

Precondiciones generales

1. Para el modelo de simulación generado, sólo será posible la replicación a nivel de tablas, no a nivel de items de datos. Esto significa que cada tabla definida en el modelo de datos de la simulación se encuentra totalmente replicada en los nodos donde se define que estará. En consecuencia, un nodo puede contener o no la réplica de una tabla. No se admiten réplicas parciales o fragmentaciones. No se admiten fragmentaciones, esto se prevé de ampliar en trabajos futuros.
2. Los nodos definidos estarán siempre conectados a la red, a fin de evitar considerar los casos de actualizaciones y propagaciones a nodos desconectados.

Etapas necesarias para generar una simulación

El usuario define un modelo donde debe especificar:

- Nombre del modelo
- N tablas, con sus respectivos campos
- M nodos o localidades
- Para cada una de las N tablas, si está replicada o no, y de estarlo, los nodos donde se replica.
- Dentro del grafo de nodos definido, el peso de comunicación entre dos nodos conectados. Esta conexión es bidireccional, de manera que se obtiene un grafo no dirigido.
- Porcentaje de operaciones a realizarse en el modelo, definiendo porcentaje de altas, bajas, modificaciones y consultas, y a su vez estos mismos porcentajes **por tabla**, ya que la realidad indica que dentro de un modelo de datos existen tablas con mayor probabilidad de actualizaciones, y otras con menor probabilidad, las cuales son solamente consultadas y raramente o nunca actualizadas. La definición del porcentaje de operaciones a nivel de tablas permiten ajustar el modelo a la realidad, y evitar tener probabilidades iguales de actualizaciones en las tablas del modelo.

- Generación de las trazas de operaciones por nodo, respetando el porcentaje de operaciones definido anteriormente.
- Ejecución de las trazas de operaciones por nodo, donde finalmente se utiliza la información definida para el modelo.
- Comparación de los valores obtenidos a través del módulo graficador de resultados.

Como es de esperar, el usuario del simulador puede querer comparar diferentes combinaciones entre distribución de los datos en los nodos y porcentajes de operaciones para el modelo. Para ello se introduce el concepto de **corrida**, de manera de poder definir varias corridas por modelo y así compararlas y elegir la óptima en la etapa de decisión.

Hay que tener en cuenta que por corrida se puede variar el porcentaje de operaciones y la forma en que se replican las tablas en los nodos, **no los nodos y sus conexiones**. Esto es lógicamente así porque como base del problema se encuentran los sitios en los cuales se va a distribuir la base de datos, y estos son físicamente estables y conectados entre sí en forma permanente.

En caso de querer evaluar las conexiones entre nodos, debe plantearse el problema como dos modelos diferentes, a fin de poder cambiar la distribución y conexiones.

Para utilizar *SimReplica*, y luego de instalarlo (guiado por un asistente), se deben definir los parámetros que se utilizarán para la generación y ejecución de las trazas. La simulación arrojará resultados, los cuales pueden ser visualizados, interpretados y analizados por los diferentes gráficos provistos por la herramienta.

4.2.- Proceso de Generación de trazas

El algoritmo generador de trazas generará N archivos de trazas, una por cada nodo, cada archivo con T trazas cada uno, donde cada traza indicará que operación será evaluada en la etapa posterior de ejecución.

Debe tenerse en cuenta:

- N y T son parámetros definidos por el usuario. N es la cantidad de nodos que componen la red (definidos en una etapa previa), y T, que corresponde a la cantidad de trazas a generar por nodo, debe ser establecido al momento de disparar la generación.
- El porcentaje de operaciones definidas para el modelo de datos, y a su vez, el porcentaje de operaciones por tabla.

La salida de este proceso serán M archivos de T trazas cada uno, y cada traza tendrá el siguiente formato:

(<tipo_de_operación>, <tabla>, <Δt>)

donde

- <tipo_de_operación> corresponde a A (Alta), B (Baja), M (Modificación) o C (Consulta)
- <tabla> corresponde a alguna de las N tablas definidas por el usuario en su modelo.
- <Δt> corresponde al tiempo que debe transcurrir entre la finalización de la ejecución de la traza corriente y el comienzo de la siguiente.

Por ejemplo, suponiendo que la cantidad de trazas T es 100, habiendo definido un 50 % de altas, 30% de modificaciones, 10% de bajas y 10% de consultas, tenemos una distribución de 50, 30, 10 y 10 trazas respectivamente. Como el modelo consta de N tablas (t1, t2, ..., tN), el 50% de altas es distribuído entre las mismas de la siguiente forma: 80% sobre t1, 5% sobre t2, 3% sobre t3, etc., lo que indicaría un total de 40 trazas de altas sobre t1, 3 trazas de altas sobre t2, 1 traza de alta sobre t3, etc.

Validaciones previas al proceso de generación:

1. Deben existir corridas definidas para el modelo.
2. Deben estar definidos los porcentajes de operaciones para el modelo/corrida ingresado.

3. Deben estar definidos los porcentajes de operaciones a nivel de tablas para el modelo/corrida.

Las dos últimas exigencias son necesarias para poder calcular el número de operaciones por tabla para el archivo de trazas que será generado.

El proceso de generación debe finalizar con la creación de un archivo, cuyo nombre tendrá el siguiente formato:

fingenMMM.CCC

donde:

- MMM: número de modelo.
- CCC: número de corrida.

Este archivo permite sincronizar los procesos Java con el programa Visual Basic que los dispara. Una vez que la aplicación VB detecta el archivo, lo elimina.

Validaciones posteriores al proceso de generación:

Verificar que todos los archivos de trazas generados para cada nodo sean de tamaño (en KB) mayor que cero. En caso que alguno sea de tamaño cero, el proceso emite el mensaje de error correspondiente y elimina todos los archivos de trazas.

En caso de existir archivos de trazas generados para el modelo/corrida, y se dispare nuevamente este proceso, los archivos de trazas generados anteriormente serán reemplazados por los nuevos.

Implementación en Java del proceso de Generación de Trazas

Se implementaron en Java las siguientes clases:

- MainGenerador
- SubMainGenerador
- GeneradorTraza
- RandomPorTabla
- OperaciónPorTabla
- Conexion
- adminContador

La relación entre estas clases se muestra gráficamente en la siguiente figura (Fig. 4.1):

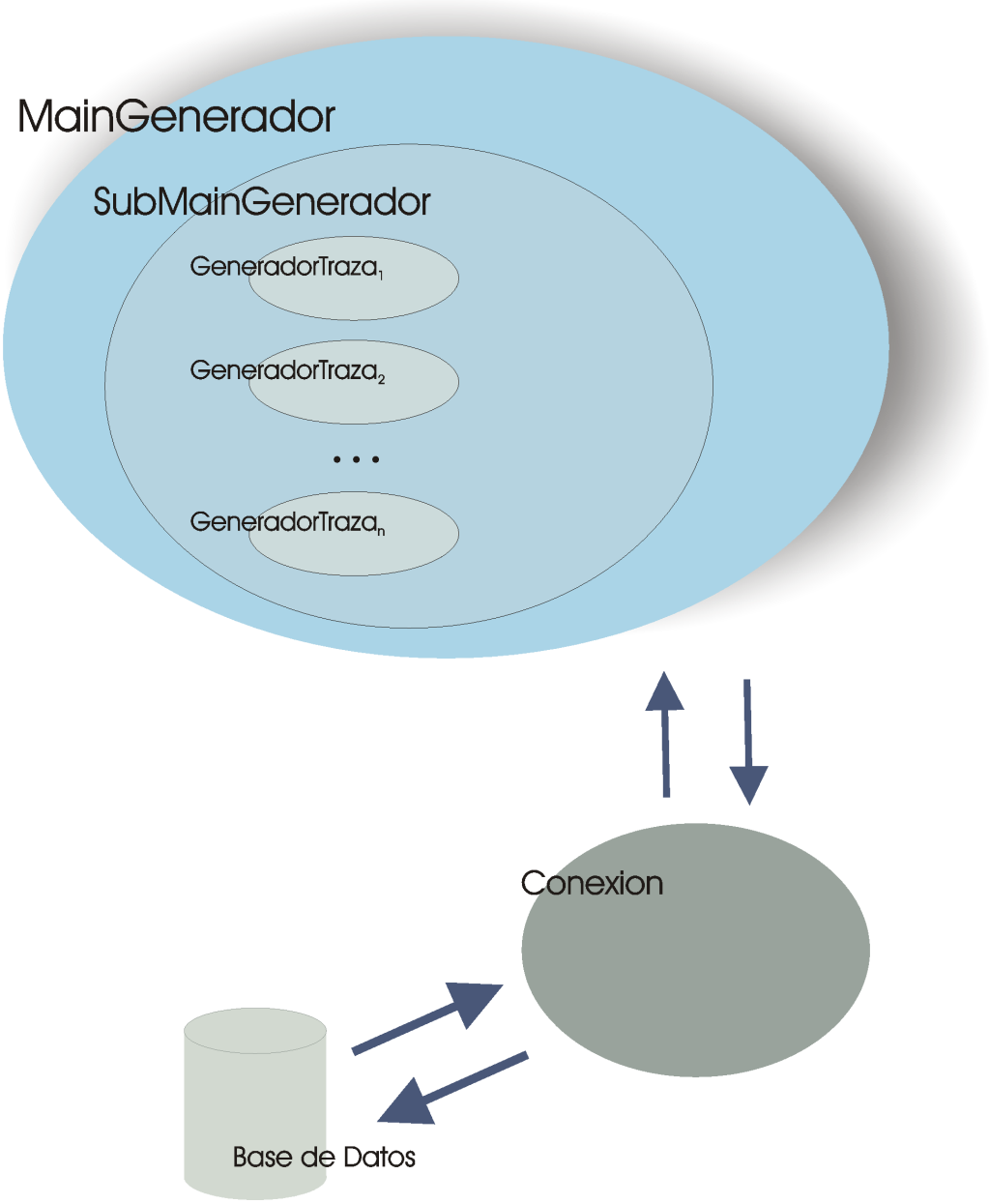


Fig. 4.1: Proceso de Generación de Trazas

- `MainGenerador`: Esta clase es simplemente la que conecta el código VB con el código Java. Actúa de interface entre estos. Solamente recibe los parámetros que VB envía para comunicarse y crea instancias de las clases que realmente harán la tarea de generar las trazas. El código (desde VB) que comienza con la generación de las trazas es:

  java MainGenerador x1 x2 x3 x4

  donde:

  - java = llamada al intérprete Java para la ejecución. Deben estar apropiadamente definidos los valores del CLASSPATH en la máquina en que se está trabajando.

- x1 = código de Modelo
- x2 = cantidad de Trazas
- x3 = código de Corrida
- x4 = path de ejecución de la aplicación

`MainGenerador` genera una instancia de la clase `SubMainGenerador` por cada ejecución.

- `SubMainGenerador`: esta clase obtiene todos los nodos que componen el modelo y procede a generar las trazas para cada uno de ellos. Esta tarea implica generar una instancia de `GeneradorTraza` para cada nodo, parametrizando con las variables necesarias en cada llamado. Una vez que finaliza, sincroniza la terminación de cada instancia de `GeneradorTraza`, a través de la utilización de una instancia de la clase `adminContador`.

  Como resultado de la finalización de la generación trazas en todos los nodos se crea un archivo llamado **FinGenMMM.CCC**, donde:

  - MMM= código de modelo
  - CCC= código de corrida

  que indica que sucedió dicho evento. La generación de este archivo permite la sincronización entre el programa VB y el código Java: si bien Java termina, VB espera hasta que ese archivo sea generado.

- `GeneradorTraza`: para un modelo, corrida y nodo dado junto al número de trazas T seteado por el usuario, se realizará el cálculo de la cantidad de operaciones que contendrá dicho nodo, es decir, la cantidad de altas , bajas, modificaciones y consultas y sobre que tabla se realizará cada operación. Para este cálculo se tienen en cuenta los porcentajes de operaciones definidos para el modelo/corrida, así como también el porcentaje de operaciones por tabla del modelo.

  El archivo de trazas generado tendrá una ordenación aleatoria y cada traza tendrá un tiempo de demora (Δt) antes del comienzo de la siguiente traza. Para lograr este efecto se instancia la clase `RandomPorTabla` la cual genera para un código de tabla dado un número aleatorio entre dos valores ya definidos.

  Para la generación del archivo de trazas se necesitará de la clase `OperacionPorTabla`, la cual almacenará la cantidad de A, B, M y C por cada tabla.

El resultado final del proceso de generación de trazas será la generación de un archivo de trazas con el siguiente nombre:

TrazaMMMNNN.CCC

donde:

- MMM= código de modelo
- NNN = código de nodo
- CCC= código de corrida

Además de las clases descriptas, el proceso de generación de trazas se complementa con las clases `adminContador` y `Conexión`. La primera es una implementación de varios contadores compartidos entre varios procesos, de manera que el uso de los mismos esté perfectamente sincronizado. La clase `Conexión` implementa el acceso a una base de datos a través del uso de ODBC. Los métodos implementados dentro de esta clase sirven para que el acceso a la base de datos sea idéntico en cada caso que sea necesario, y para abstraer el acceso a los datos a través de métodos puntuales y fácilmente utilizables.

4.3.- Proceso de Ejecución de trazas

El proceso de ejecución de trazas consiste en simular una ejecución real en los nodos o sitios de la red distribuída, donde las operaciones serán virtualmente realizadas en la base de datos distribuída. Estas operaciones estarán indicadas en las trazas generadas para cada nodo en la etapa de generación de trazas.

SimReplica simula la ejecución de trazas implementando el protocolo de replicación Lazy – Master.

La ejecución de las trazas en un nodo está compuesta por las siguientes tareas:

1. Identificar el tipo de operación para determinar los pasos a seguir.
2. Medir la simulación de la ejecución. Esto depende de:
    - Tipo de operación, para determinar los tiempos de ejecución involucrados, a partir de valores tabulados.
    - Tiempo de localización de los datos, en función de la tabla donde se va a realizar la operación, y de los nodos en los cuales se ha replicado la misma.
    - Tiempo de actualización de las réplicas, lo cual depende del algoritmo de replicación elegido, y de los tiempos involucrados en el mismo.

Si el tipo de operación es Consulta, deberá considerarse el camino más corto dentro del grafo de nodos para recuperar el dato. El camino más corto se obtiene haciendo uso de los pesos de comunicación entre los nodos.

Si el tipo de operación es una actualización (Alta, Baja o Modificación) siempre se recurre al nodo propietario o master de la tabla, el cuál es el encargado de realizar y coordinar la actualización requerida (debido al protocolo elegido). La comunicación entre el nodo solicitante y el nodo master será sincrónica: el nodo solicitante espera hasta que el master le envíe el acknowledge acusando que ha culminado la atención del pedido, lo cuál no significa que la actualización de las réplicas se realicen en ese instante de tiempo. Cuando la master finaliza, el nodo solicitante da por finalizada la ejecución de esa operación. Luego la master dispara la actualización de todas las copias que no participaron de la transacción original. El tiempo demorado en la actualización de las réplicas se carga al tiempo total de ejecución del nodo master.

Altas	Son todas consideradas "actualizaciones". El tiempo de ejecución de las mismas incluye el tiempo de actualización de réplicas, tarea que llevará a cabo el nodo master.
Bajas	
Modificaciones	
Consultas	Solo implica la localización de los datos, usando para ello un algoritmo para encontrar el camino más corto.

Validaciones previas:

1. El proceso VB debe validar la existencia de los archivos de trazas para cada nodo, teniendo en cuenta el modelo y la corrida. Entonces, para cada nodo, debe validar la existencia del archivo:

   TrazaMMMNNN.CCC

   donde:

   - MMM: número de modelo.
   - NNN: número de nodo.
   - CCC: número de corridas.

   (Todos estos valores están rellenos con ceros a izquierda).

   Este archivo debería haber sido generado por el proceso de generación de Trazas.
2. El proceso VB debe verificar que para cada una de las tablas del modelo seleccionado exista un nodo del mismo propietario de la misma. En caso contrario no debe permitirse la ejecución.

Validaciones posteriores a la ejecución:

1. La ejecución de trazas (proceso Java) debe finalizar generando el archivo:

   finMMM.CCC

   donde:

   - MMM: número de modelo
   - CCC: número de corridas

   La generación de este archivo implica la sincronización entre el proceso Java y la aplicación VB. Una vez que la aplicación VB lo detecta, debe eliminarlo.
2. Verificar que para cada nodo del modelo/corrida se haya generado un resultado distinto de cero. En caso de existir un resultado para un nodo igual a cero, se emite el mensaje de error correspondiente y se eliminan todos los resultados obtenidos de ese modelo/corrida.

Descripción general del proceso de ejecución de una traza

Caso 1: Operaciones Alta, Baja y Modificación

Pueden considerarse dos subcasos:

1. Si existe la tabla en el nodo y éste es propietario de la misma, entonces se ejecuta la operación. Luego se actualizan los datos en la réplicas. Los tiempos involucrados en este caso serán: tiempo de actualización (teniendo en cuenta el tipo de operación y la cantidad de bytes que ocupa un registro de la tabla), y el tiempo de actualización de cada réplica.
2. El nodo *NO* es propietario de la tabla, entonces se debe pedir la actualización al nodo propietario de la misma. Los tiempos involucrados serán el tiempo de sincronización con el propietario de la tabla, encargado de ejecutar la actualización real. Una vez que el nodo master actualiza la copia master, envía el acknowledge al nodo solicitante. En este momento se da por finalizada la tarea para el nodo solicitante. El nodo master debe continuar la tarea de actualización de cada réplica, incrementando con estos valores el tiempo propio de procesamiento.

Caso 2: Operación Consulta

También pueden analizarse dos subcasos:

1. La tabla que se desea consultar está replicada en el nodo solicitante de la operación (sin importar si es propietario o no de la misma). El tiempo de la consulta es mínimo, involucrando en este caso el tiempo de consulta, teniendo en cuenta además la cantidad de bytes a leer.
2. La tabla no se encuentra replicada en el nodo solicitante. En este caso se busca el camino más corto dentro de la red de nodos. Los tiempos involucrados en este caso serán el tiempo de consulta del registro (considerando la cantidad de bytes que ocupa un registro de la tabla) ponderando este tiempo con el peso obtenido al encontrar el camino más corto hasta el dato.

Los tiempos tabulados son:

- Tiempo de operación ALTA en una tabla
- Tiempo de operación MODIFICACION en una tabla
- Tiempo de operación BAJA en una tabla
- Tiempo de CONSULTA de datos de una tabla
- Tiempo de actualización de una réplica

Se mencionó el Tiempo de sincronización con el nodo propietario de la tabla. Este tiempo se calcula en ejecución, no se encuentra tabulado como los casos anteriores.

Los tiempos de ALTA, BAJA, MODIFICACION y CONSULTA se tabularán para 1 byte de datos. Este deberá ser multiplicado por la cantidad de bytes que ocupe el registro a dar de alta, modificar, eliminar o consultar.

Estados posibles de un nodo

- Actualizando datos propios que surgen de una traza propia.
- Actualizando datos propios que surgen de una traza ajena.
- Actualizando réplicas.
- Requiriendo actualización de datos al nodo master.
- Consultando datos de los cuáles posee réplicas (sin importar la propiedad de los mismos).
- Consultando datos de los cuáles no posee réplicas.

Todas estas consideraciones son tenidas en cuenta al momento de implementar en Java el proceso de ejecución de trazas.

Implementación en Java del proceso de Ejecución de Trazas

Se implementaron en Java las siguientes clases:

- MainPpal
- MainEjecutador
- AdminNodo
- NodoEnEjecucion
- elemenPedExt
- atencionExterna
- listPedidosExt
- TamaniosTablas
- TrazaEnEjecucion
- InfoXTabla
- adminContador
- adminGrafo
- Edge
- Node
- Graph
- DistanciasTablas
- elemenCamino
- Conexion

La relación entre las clases principales (aquellas que representan procesos) se muestra en la siguiente figura (Fig. 4.2):

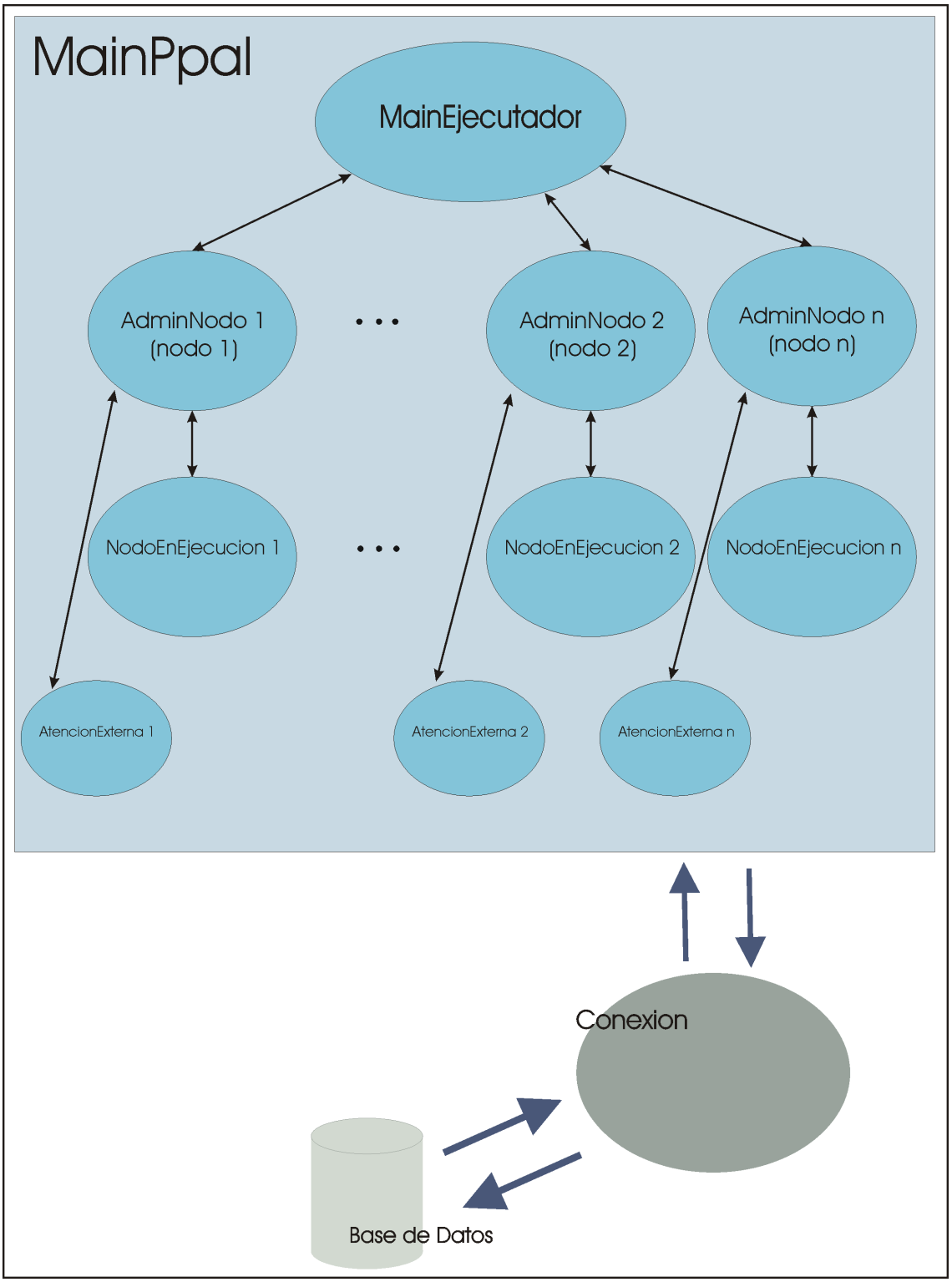


Fig. 4.2: Proceso de Ejecución de Trazas

- `MainPpal`: Esta clase es la que conecta el código VB con el código Java. Actúa de interface entre estos. Recibe los parámetros que VB envía para comunicarse y crea instancias de las clases que realmente harán la tarea de ejecutar las trazas. El código (desde VB) que comienza con la ejecución de las trazas es:

  java MainPpal x1 x2 x3

donde:

- java = llamada al intérprete Java para la ejecución. Deben estar apropiadamente definidos los valores del CLASSPATH en la máquina en que se está trabajando.
- x1 = código de Modelo
- x2 = código de Corrida
- x3 = path de ejecución de la aplicación

Esta clase instancia la clase `MainEjecutador`, la cuál utilizará los valores de dichas variables.

- `MainEjecutador`: este proceso genera M instancias de la clase `AdminNodo` (una por cada nodo del modelo), clase encargada de administrar cada nodo, enviando a cada una los seteos del modelo. Además dispara la ejecución de dichos administradores.

  Por otro lado, `MainEjecutador` se encarga de administrar aquellas tareas que los nodos no pueden resolver por falta de acceso a las tablas, es por ello que éstos se lo envían a su respectivo administrador y de allí al `MainEjecutador`, el cual redistribuye la tarea al nodo que sí puede trabajar con la tabla y resolver la operación. Por ejemplo, cuando un nodo tiene que actualizar una tabla de la cuál no es propietario, envía la tarea a su administrador, y este a `MainEjecutador`, el cuál mantiene una lista de pedidos externos. Cada administrador, cuando está ocioso, toma una de esas tareas, y la atiende (instanciando la clase `atencionExterna`).

  Además, `MainEjecutador` coordina la finalización de cada administrador de nodo y la culminación de tareas pendientes a redistribuir.

  Al finalizar la ejecución de todas las trazas, actualiza los resultados de dicho trabajo (en la base de datos) y se genera un archivo denominado FinMMM.CCC donde:

  - MMM= código de modelo
  - CCC= código de corrida

  el cual señala la finalización general de la ejecución de las trazas. La generación de este archivo es solamente para coordinar el proceso Java con la ejecución de VB.

- `AdminNodo`: esta clase se encarga de administrar cada nodo del modelo, tanto en cuanto a su ejecución local como en la atención de pedidos externos. Por un lado, da la señal de comienzo a su respectivo nodo (se instancia la clase

NodoEnEjecucion), mientras que por otro redirecciona los pedidos de actualización de tablas de las cuales el nodo no es propietario. En este caso, AdminNodo actúa de puente a MainEjecutador, el cuál organiza los pedidos de atención externa, y los envía al administrador correspondiente. Además, de MainEjecutador recibe los pedidos de atención externa generados por otros nodos y que deben ser atendidos por el nodo que administra, debido a que este es propietario de la tabla en la cuál se realizará la actualización. En este caso se crea una instancia de la clase atencionExterna, que es la encargada específica de atenderlo.

- NodoEnEjecucion: esta clase es instanciada por AdminNodo. Se encarga puntualmente de recorrer el archivo de trazas del nodo, y procesar cada una de las trazas. En caso que la traza corresponda a una actualización (alta, baja o modificación) el nodo determina si es propietario o no de la tabla sobre la cuál se realizará. Si es propietario, calcula los tiempos correpondientes a la actualización, y actualiza sus contadores de tiempo. En caso de no ser propietario, envía el pedido de actualización a su administrador, el cuál operará oportunamente. Si la traza corresponde a una consulta, determina si tiene la tabla replicada (sin importar si es o no propietario de la misma). En caso de tenerla, actualiza los tiempos en función de la consulta, mientras que si no la tiene pide a su administrador que resuelva el camino más corto a alguna réplica de la tabla en toda la red de nodos, y en función de esta respuesta actualiza sus contadores de tiempo.
- atencionExterna: esta clase es instanciada (un número indeterminado de veces) por AdminNodo. Es la encargada de atender los pedidos externos del nodo al que representa. Una vez finalizada la atención, la instancia desaparece terminando la ejecución. Solamente se encarga de calcular el tiempo de actualización de la tabla pedida (discriminando si es alta, baja o modificación), sumado al tiempo de actualización de todas las réplicas de dicha tabla. Una vez que finaliza, avisa a la instancia de AdminNodo que lo creo que ha finalizado.

Las demás clases sirven de complemento para toda la tarea de simulación. Conexión es exactamente igual a la clase usada y descripta en la generación de Trazas. Lo mismo ocurre con adminContador. Las clases Node, Edge, Graph y adminGrafo trabajan en conjunto, y el objetivo de las mismas es armar la red de nodos y obtener el

camino más corto desde un nodo dado a una tabla puntual. En cuanto al algoritmo de búsqueda del camino mínimo, este no responde a ninguno conocido ya que el problema en este caso no se asemejaba a ninguno de ellos. Se tomaron las ideas de cada uno que servían en esta implementación y se ensamblaron obteniendo un resultado adecuado al problema.

Las demás clases no mencionadas explícitamente, si bien para el conjunto del proceso son importantes, particularmente cada una de ellas no tiene una función relevante, con lo cuál se obvia su desarrollo detallado.

Capítulo 5: Análisis y evaluación de resultados

Introducción

En el capítulo anterior se describió en detalle la forma en que *SimReplica* genera y ejecuta las trazas, destacando los procesos principales involucrados en estas tareas.

Al finalizar la ejecución de cada traza en cada nodo, el resultado será un número correspondiente al tiempo total transcurrido desde el comienzo hasta la finalización de la ejecución de la traza.

Los resultados obtenidos serán analizados por corrida, y como se indicó anteriormente, por modelo es posible definir diferentes corridas, las cuales están relacionadas con el porcentaje de operaciones y la forma de replicación de las tablas en los nodos o localidades del modelo. Pueden definirse diferentes corridas para el modelo variando estos dos parámetros (operaciones y esquema de replicación), dando lugar a la posibilidad de comparar distintas variantes para un mismo modelo. Cabe recordar además que las trazas se generan en función de estos parámetros mencionados.

Los resultados obtenidos pueden analizarse con la herramienta implementada, o en forma manual accediendo a los datos almacenados en la base de datos.

5.1.- Análisis de resultados utilizando *SimReplica*

SimReplica permite mostrar los resultados obtenidos de la ejecución de las trazas a través de diferentes gráficos. No necesariamente los cuatro gráficos pueden servir para el análisis en todos los casos, la elección de cada uno para análisis debe realizarse de manera tal que represente adecuadamente la información necesaria.

El *primer gráfico* permite visualizar para un modelo/corrida, a través de un gráfico de barras, la relación Nodo – Tiempo de ejecución de trazas. Es sencillo, se limita a mostrar los resultados obtenidos por nodo. No compara los resultados entre las corridas de un modelo.

El *segundo gráfico*, también a través de un gráfico de barras, muestra la comparativa entre corridas para cada nodo de un mismo modelo. Por ejemplo, habiendo obtenido los siguientes valores:

	Corrida 1	**Corrida 2**	**Corrida 3**
Nodo 1	10	12	20
Nodo 2	12	12	21
Nodo 3	15	15	15

Se mostrará el siguiente gráfico (Fig. 5.1):

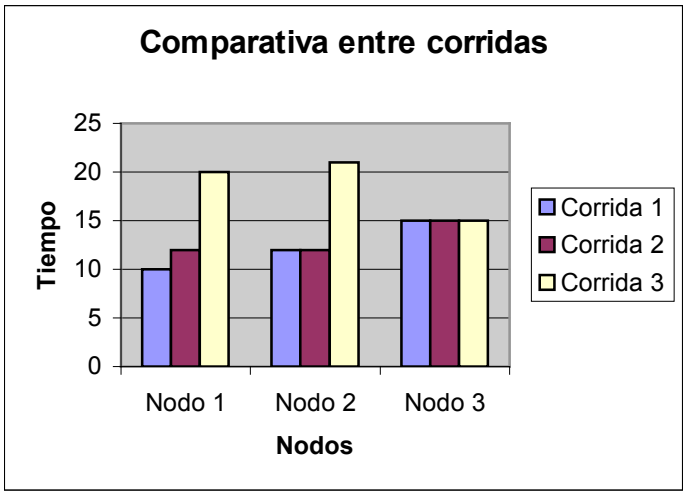


Fig. 5.1: Estilo del gráfico de comparativa entre corridas

Es útil para visualizar los tiempos de respuesta cuando en las corridas se ha variado algún parámetro, como por ejemplo el porcentaje de replicación o el porcentaje de operaciones.

El *tercer gráfico* que provee *SimReplica* es una *Proyección fijando el esquema de replicación y variando el porcentaje de operaciones*. Se vale de un gráfico de líneas, donde el eje X representa el porcentaje de actualizaciones, y el eje Y el tiempo promedio de respuesta por corrida. Para que este gráfico sea de utilidad, deberán existir corridas donde el porcentaje de actualizaciones varíe de 0 a 100 para cada caso. Al considerar actualizaciones, se consideran tanto altas, como bajas y modificaciones.

Por ejemplo, habiendo obtenido los siguientes valores:

Corrida	Valor Promedio	Actualizaciones	Consultas
Corrida 1	10	0	100
Corrida 2	16	10	90
Corrida 3	17	20	80
Corrida 4	22	30	70
Corrida 5	25	40	60
Corrida 6	30	50	50
Corrida 7	45	60	40
Corrida 8	50	70	30
Corrida 9	52	80	20
Corrida 10	59	90	10
Corrida 11	61	100	0

Se mostrará el siguiente gráfico (Fig 5.2):

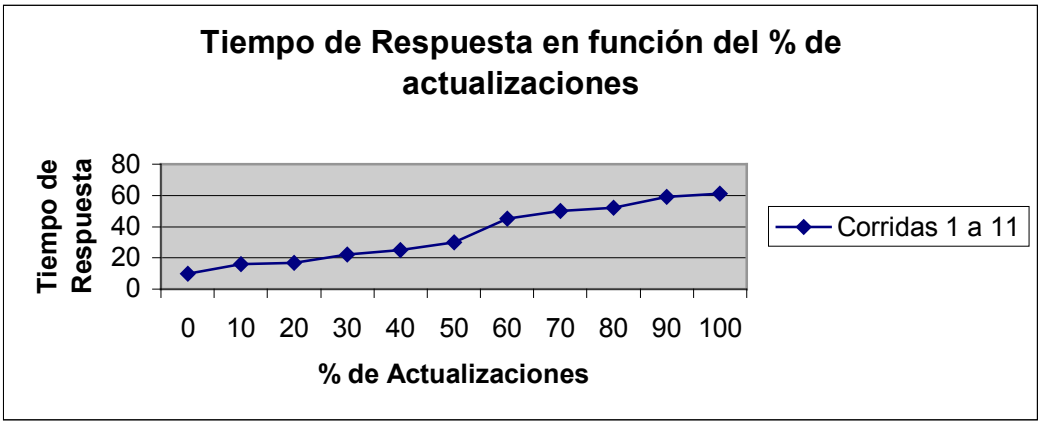


Fig. 5.2: Estilo del tercer gráfico provisto por *SimReplica*

El valor promedio mencionado en la tabla del ejemplo se calcula promediando los tiempos de todos los nodos de una corrida dada.

Este gráfico permitirá analizar el rendimiento del modelo en función del esquema de réplicas definido.

Tiene sentido cuando se definen más de dos corridas, variando el porcentaje de operaciones del modelo en cada corrida.

El *cuarto y último gráfico* permite obtener una *Proyección fijando el porcentaje de operaciones y variando el esquema de replicación.* Deberá evaluarse el porcentaje de replicación del modelo por corrida (de 0 a 100%), valor que conformará el eje X, y los valores en Y serán el promedio de tiempo de procesamiento de todos los nodos. También estará representado a través de un gráfico de líneas. La cantidad de valores sobre el eje X será variable, en función de la cantidad de corridas definidas.

Para calcular el grado o porcentaje de replicación que se asocia a una corrida, se procede de la siguiente forma:

1. Determinar el porcentaje de replicación de cada tabla del modelo/corrida, denotado por R_n, con n = 1 .. <Total de tablas del modelo>.
2. Determinar el porcentaje de replicación del modelo/corrida, en función del valor obtenido en 1), denotado por R.

R_n, se define como sigue:

$$R_n = \frac{N(T_n) * 100}{N}$$

Donde

T_n = Tabla n del modelo (n = 1 .. Total de tablas del modelo)

N = Cantidad total de nodos del modelo

N (T_n) = Cantidad de Nodos del modelo en que se replica T_n (incluyendo el nodo propietario).

Finalmente, para obtener R se establece la siguiente relación:

$$R = PROM(R_1, R_2, ..., R_n)$$

$$R = \frac{R_1 + R_2 + ... + R_n}{N}$$

donde N se define de la misma forma que en el caso anterior.

5.2.- Experimentación: casos de prueba evaluados

Para poder evaluar y medir la performace del sistema y procesos implementados, se definieron cuatro modelos, todos básicamente iguales, con 7 tablas, 10 nodos o localidades y sus respectivas interconexiones. Se dividieron las pruebas en dos etapas, trabajando con dos modelos en la primera y con otros dos en la segunda.

En la primer etapa se intentó evaluar la performance del proceso y el comportamiento general del modelo fijando el porcentaje de operaciones y variando el esquema de replicación.

En la segunda etapa se fijó el esquema de replicación, y se varió el porcentaje de operaciones asociado.

A continuación se detallan ambas etapas.

Primer Etapa

Se trabajo con dos modelos, que en adelante llamaremos *Modelo 1* y *Modelo 2*. En los dos casos se fijó el porcentaje de operaciones y se varió el esquema de replicación. En el *Modelo 1* se estableció una preponderancia de actualizaciones (altas, bajas y modificaciones) que alcanzan el 90%, mientras que en el *Modelo 2* se intensificaron las consultas, llegando al 70% del total de las operaciones por nodo.

Se definieron en los dos casos cinco corridas para cada modelo, donde se fue variando el esquema de replicación de las tablas en los nodos o localidades. Para los dos modelos fueron iguales, como se indica en la siguiente tabla:

Corrida	Porcentaje de replicación general	Descripción general
1	10%	Esquema totalmente centralizado. Un nodo tiene todas las tablas y es propietario de las mismas.
2	20%	Dos nodos tienen todas las tablas replicadas, y son propietarios de las mismas.
3	30%	Dos nodos son propietarios de todas las tablas, y una más tienen réplicas de las mismas.
5	45%	Dos nodos son propietarios de las mismas, y cuatro más tienen réplicas de algunas de las mismas.
4	100%	Dos nodos son propietarios de todas las tablas, y el resto tiene réplicas de todas. Este es un esquema totalmente replicado.

La definición específica de ambos modelos se detalla en el Apéndice A.

En el primer modelo, sobre un total de 10.000 trazas de operaciones por nodo, se obtuvieron los resultados que se muestran en el siguiente gráfico (Fig. 5.3):

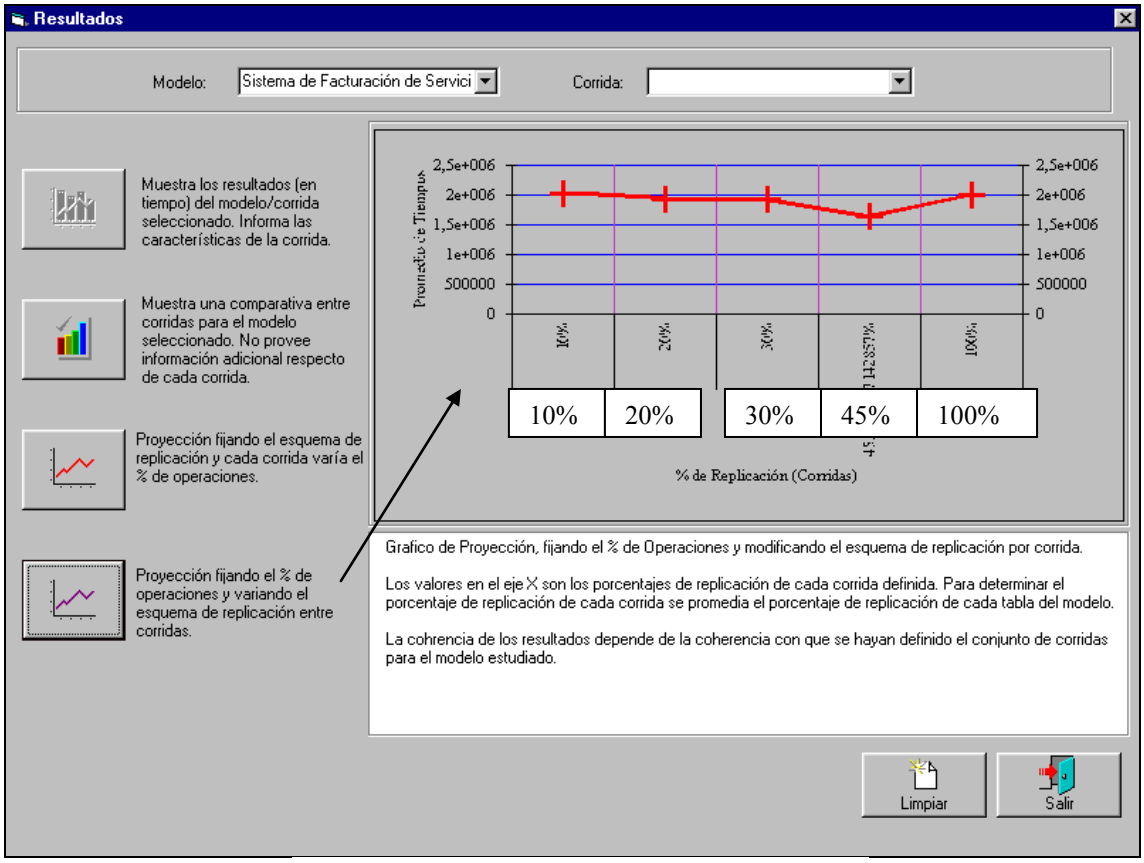


Fig. 5.3: Curva obtenida con el *Modelo 1*

El gráfico fue obtenido de *SimReplica*, se utilizó la opción de *Proyección fijando el porcentaje de operaciones*. Cada porcentaje en el eje X representa el grado de replicación de cada corrida, en particular en el siguiente orden: Corrida #1, 10%, Corrida #2, 20%, Corrida #3, 30%, Corrida #5, 45% y Corrida #4, 100%.

La curva obtenida muestra que a mayor grado de replicación, mejora el tiempo de procesamiento por nodo, exceptuando el caso de replicación total (100% de replicación), ya que al existir preponderancia de operaciones de actualización el hecho de tener todo replicado no ayuda a mejorar la performance general.

La mejoría en la performance general se nota aumentando el grado de replicación hasta un 45% (en el caso del ejemplo evaluado), y empeorando a medida que nos acercamos al 100%. Ahora bien, este caso nos hizo pensar si realmente el percentil inferior en la curva de la Fig. 5.3 era una replicación del 45%. En consecuencia se rediseñó el caso de prueba agregando dos corridas con porcentajes de replicación intermedias entre 45% y

100%: Corrida #6, con un porcentaje de replicación del 66% y Corrida #7, con un porcentaje de replicación general del 81%. Se generaron nuevamente los archivos de trazas para repetir la simulación, ahora de 1.000 operaciones cada una, y se obtuvieron los siguientes resultados (Fig. 5.4):

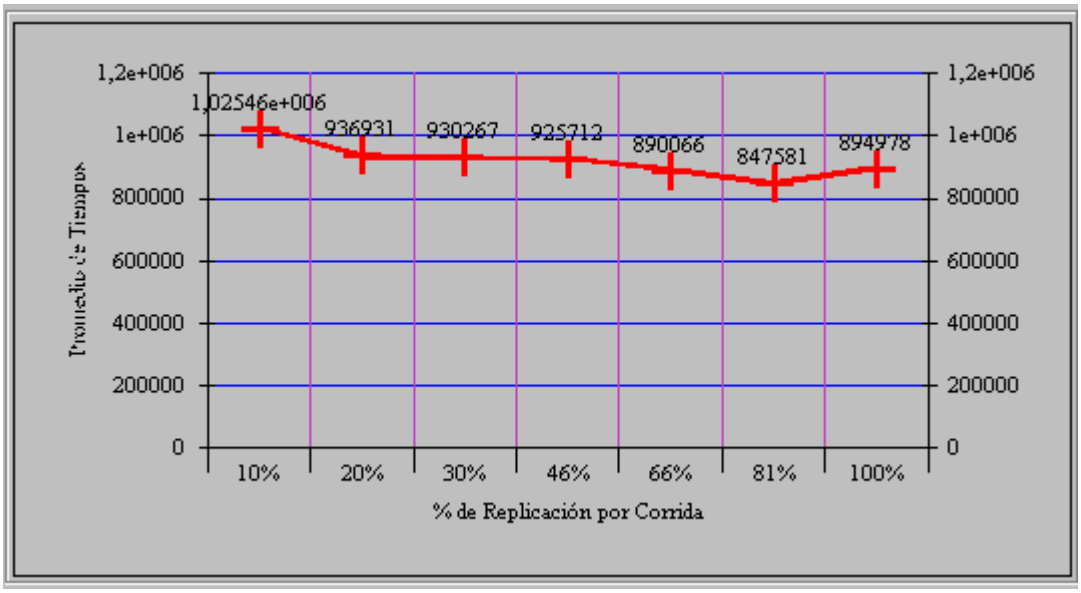


Fig. 5.4: Curva obtenida con el Modelo 1 con 7 corridas

Como puede observarse en la nueva curva obtenida, el percentil inferior está acotado en un porcentaje de replicación entre el 81% y el 100%. Esta cota es bastante mejor que la obtenida en la prueba anterior, donde el percentil inferior estaba acotado entre el 45% y 100% de replicación.

En el *Modelo 2*, sobre un total de 2.000 trazas de operaciones por nodo, se obtuvieron los resultados que se muestran en el siguiente gráfico (también obtenido de *SimReplica*, Fig. 5.5):

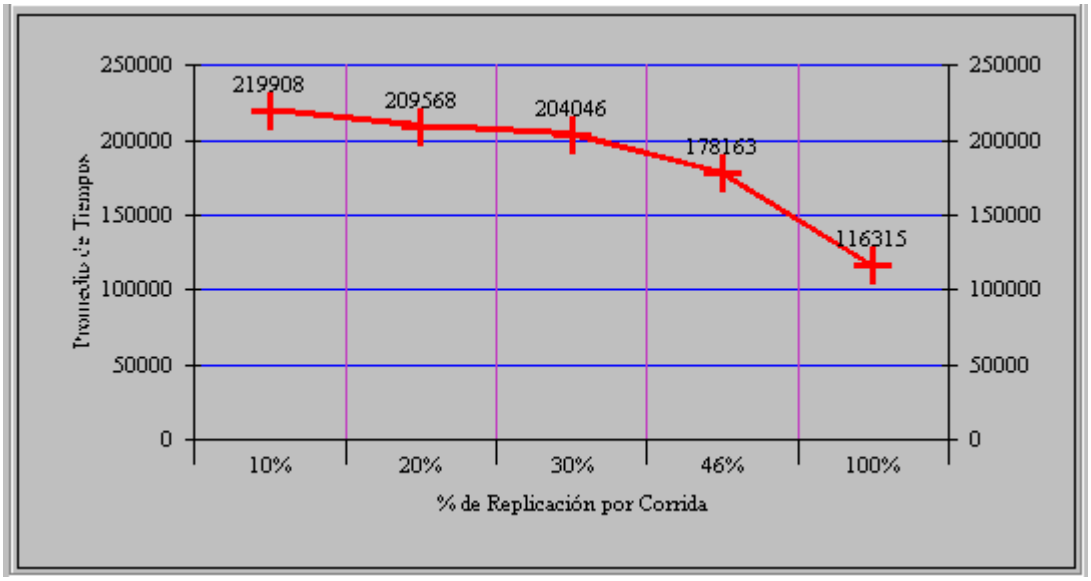


Fig. 5.5: Curva obtenida con el *Modelo 2*

En este caso el hecho de aumentar el grado o porcentaje de replicación hace que la performance general mejore debido a la preponderancia de operaciones de consulta en todo el modelo. Claramente el gráfico arroja valores satisfactorios y una curva que posee el movimiento esperado de acuerdo a las suposiciones teóricas previas en función del algoritmo implementado.

Segunda Etapa

En esta etapa se trabajó con los modelos que en adelante se identificarán como *Modelo 3* y *Modelo 4*. En los dos casos se fijó el esquema de replicación y se variaron, a través de 5 corridas para cada uno, el porcentaje de operaciones, de acuerdo a la siguiente tabla:

Corrida	**Porcentaje de Actualizaciones**	**Descripción general**
1	30%	Mayoría de Consultas, que alcanzan el 70% de las operaciones.
2	45%	Las actualizaciones se acercan a las consultas, sin alcanzarlas.
3	60%	Las actualizaciones apenas superan las consultas.
4	75%	Hay preponderancia de actualizaciones.
5	90%	El nivel de consultas es mínimo.

En cuanto al *Modelo 3*, si bien no es centralizado, tiene bajo nivel de réplicas, con todas las tablas concentradas y replicadas en sólo dos de los diez nodos del modelo. Los dos nodos se reparten la propiedad de las tablas.

La simulación sobre 1.000 trazas de operaciones en cada nodo arrojó el siguiente gráfico (obtenido de *SimReplica*, Fig. 5.6):

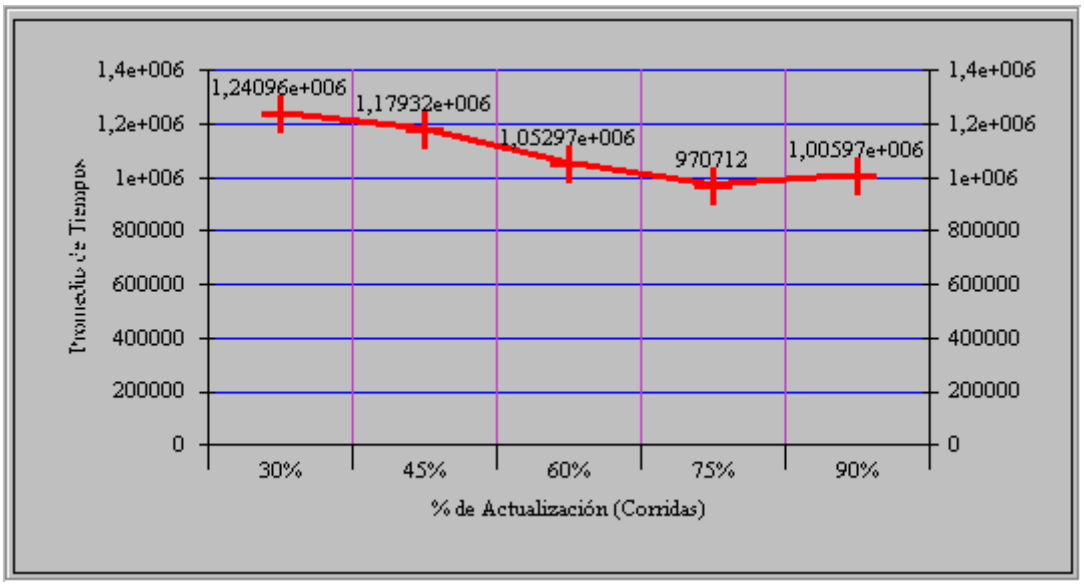


Fig. 5.6: Curva obtenida con el *Modelo 3*

Con un esquema altamente centralizado era de esperar que ante una demanda de operaciones donde la mayoría son consultas los tiempos de respuesta generales sean mayores que en un modelo donde las actualizaciones son mayoría. El gráfico utilizado para la visualización de los resultados fue el de *Proyección fijando del esquema de replicación y variando el porcentaje de operaciones* en cada corrida.

Finalmente, con respecto al *Modelo 4*, se estableció un esquema de replicación de los datos altamente distribuído, de manera de evaluar la situación opuesta a la planteada en el *Modelo 3*. Todos los nodos tienen réplicas de todas o algunas de las tablas, y en cada corrida se varía el porcentaje de actualizaciones.

La simulación sobre 1.000 trazas de operaciones en cada nodo arrojó el siguiente gráfico (obtenido de *SimReplica*, Fig. 5.7):

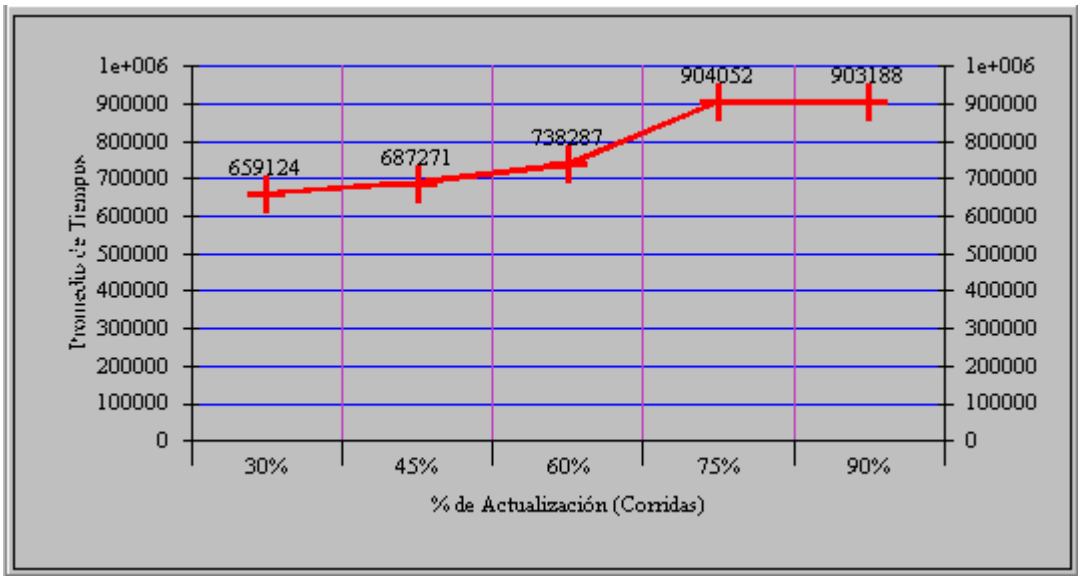


Fig. 5.7: Curva obtenida con el *Modelo 4*

Como puede observarse en el movimiento de la curva, en un esquema totalmente replicado (aunque con sólo dos nodos propietarios de las tablas) la preponderancia de consultas mejora el tiempo de respuesta de procesamiento. A medida que las consultas se hacen menores en porcentaje y aumenta la cantidad de actualizaciones, el tiempo de respuesta aumenta debido a la propagación de las actualizaciones a todos los nodos que poseen las réplicas.

En cuanto al análisis individual por corrida para el *Modelo 4*, los mayores tiempos de respuesta los arroja el nodo denominado "La Plata", ya que es propietario de las tablas con más probabilidad de ser actualizadas. Si bien el nodo denominado "Capital" también es propietario de algunas tablas, no son estas las más frecuentemente actualizadas. A modo de ejemplo, se muestra el gráfico generado por *SimReplica* (Fig. 5.8) para la Corrida #1 (para el resto de las corridas, a pesar de que difieren en los tiempos demorados por nodo, el aspecto del gráfico es similar):

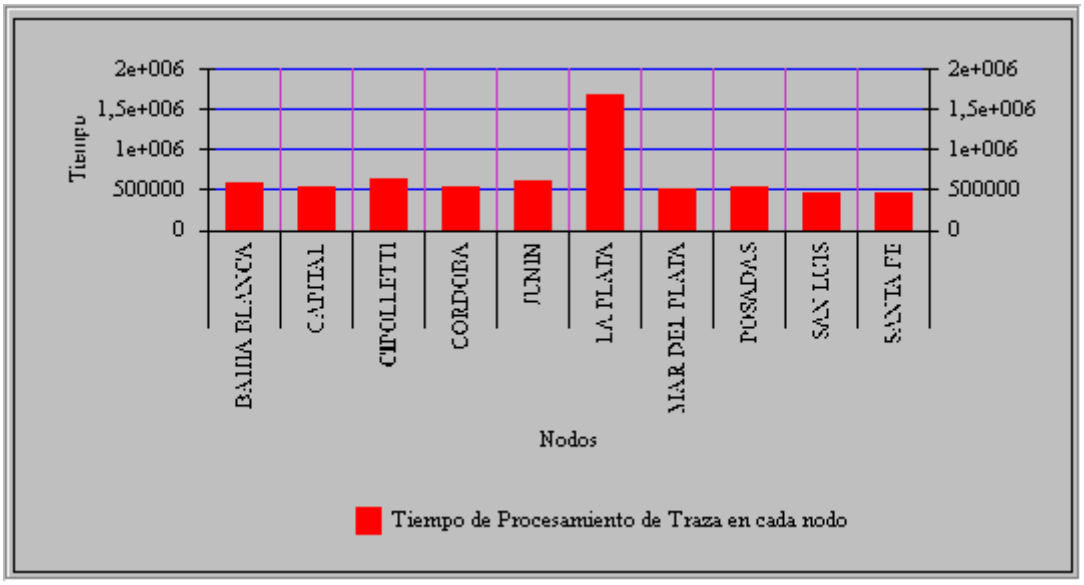


Fig.5.8: Valores obtenidos en la Corrida # 1 del *Modelo 4*

Conclusiones generales

Los cuatro casos de prueba presentados muestran los resultados esperados en función del análisis teórico de los mismos y del algoritmo de replicación implementado. Las combinaciones elegidas para esquemas de replicación y porcentaje de operaciones del modelo/corrida tratan de ser representativas en cuanto espectro de posibilidades. *SimReplica* ofrece la posibilidad de combinar el esquema de replicación y el porcentaje de operaciones por modelo/corrida de manera de poder realizar un análisis más exacto en función de los resultados que se deseen obtener. Sería una tarea muy larga detallar todos los casos, motivo por el cuál el análisis se basó en los cuatro planteados.

Capítulo 6: Conclusiones

Resultados obtenidos

Cuando se comienza a trabajar en este proyecto, el objetivo era obtener una herramienta amigable, que permitiese ayudar en la etapa de diseño de una BDD, evaluando en forma previa la distribución de los datos en función de la carga operativa del sistema.

Se pensó además en asistir la tarea de diseño de una base de datos, proveyendo la mencionada herramienta, en la que se pusiera énfasis en el análisis general de la performance del sistema a partir de la distribución de los datos y de la carga operativa. Para ello, se implementaron los procesos necesarios para permitir la simulación de la generación y ejecución de operaciones, basándose en un protocolo de replicación lazy-master, elegido éste a raíz de su auge en la mayoría de los sistemas comerciales existentes en la actualidad, no sin desconocer sus desventajas, pero omitiendo la solución a los problemas que este genera en la tarea de simulación.

El resultado principal de este trabajo de grado es *SimReplica*, una buena herramienta con interfaz visual para ingresar y presentar un modelo de datos distribuido sobre los cuañes se puede realizar una simulación de comportamiento. Además posee la flexibilidad necesaria para que, variando pocos parámetros, se puedan evaluar los resultados en forma rápida, precisa y amigable, evitando la implementación prematura de la base de datos en estudio, y postergando esta tarea hasta haber agotado una serie de posibilidades en una etapa de simulación.

Contribuciones

Las principales contribuciones de este trabajo de grado son:

- ***SimReplica*:**

  Se implementó un sistema cliente-servidor que sirve de herramienta en la etapa de diseño de una BDD, abstrayéndose en todo momento de la base de datos comercial que la implementará en una etapa posterior, haciendo hincapié solamente en la distribución de los datos y la carga operativa del modelo. Además provee el análisis de los datos obtenidos en la simulación a partir de la proyección de los mismos en diferentes tipos de gráficos.

- ***Implementación del protocolo de replicación Lazy-Master:***

  Se implementó este protocolo a fin de poder ser utilizado por *SimReplica* al momento de realizar la simulación para un modelo dado.

Trabajos Futuros

SimReplica es una herramienta elaborada con la flexibilidad suficiente para que puedan incorporarse los siguientes cambios:

1. Modificar los algoritmos para permitir la replicación a nivel de objetos de datos (actualmente se maneja a nivel de tablas), y permitir la fragmentación de los datos.
2. Elección del algoritmo de replicación para el modelo/corrida: es fácilmente implementable, ya que los procesos de simulación pueden sustituirse o elegirse, cambiando así el algoritmo al momento de simular.
3. Incorporación de mecanismos que mantengan la consistencia de los datos, en función de los algoritmos de replicación elegidos. Esto sería ideal para salvar las desventajas de cada algoritmo de replicación.
4. Incorporación de otros gráficos y herramientas para el análisis de los resultados. En función de los algoritmos de replicación implementados incorporar las herramientas necesarias para cubrir todos los espectros de análisis posibles.

Todos estos cambios mencionados seguramente incrementarán la fidelidad de la simulación, haciendo tender esta a casos implementables en las diferentes alternativas de bases de datos comerciales existentes.

Apendice A: Casos de prueba

Se desarrollaron cuatro variantes sobre un modelo de datos con el fin de obtener las mediciones y de allí en más concluir acerca de la performance del algoritmo implementado. Todas las variantes, denominadas *Modelo 1*, *Modelo 2*, *Modelo 3* y *Modelo 4* parten de la misma definición de un modelo, con idéntica cantidad de tablas, nodos o localidades, y conexiones entre ellos. En cada variante se modificó el porcentaje de operaciones o el esquema de replicación, marcando estas variaciones a partir de la definición de corridas para cada uno.

Las pruebas se organizaron en dos etapas, donde en la primera se intenta evaluar la performance del proceso y el comportamiento general del modelo fijando el porcentaje de operaciones y variando el esquema de replicación.

En la segunda etapa se fijó el esquema de replicación, y se varió el porcentaje de operaciones asociado.

Tanto la definición de los modelos como de las corridas se hicieron utilizando *SimReplica*.

A continuación se detallan los datos establecidos para el modelo utilizado en las simulaciones. Además se describen las características de cada una de las corridas.

Descripción general del Modelo

Denominación del Modelo:

Sistema de Facturación de Servicios de Internet

Tablas y Campos:

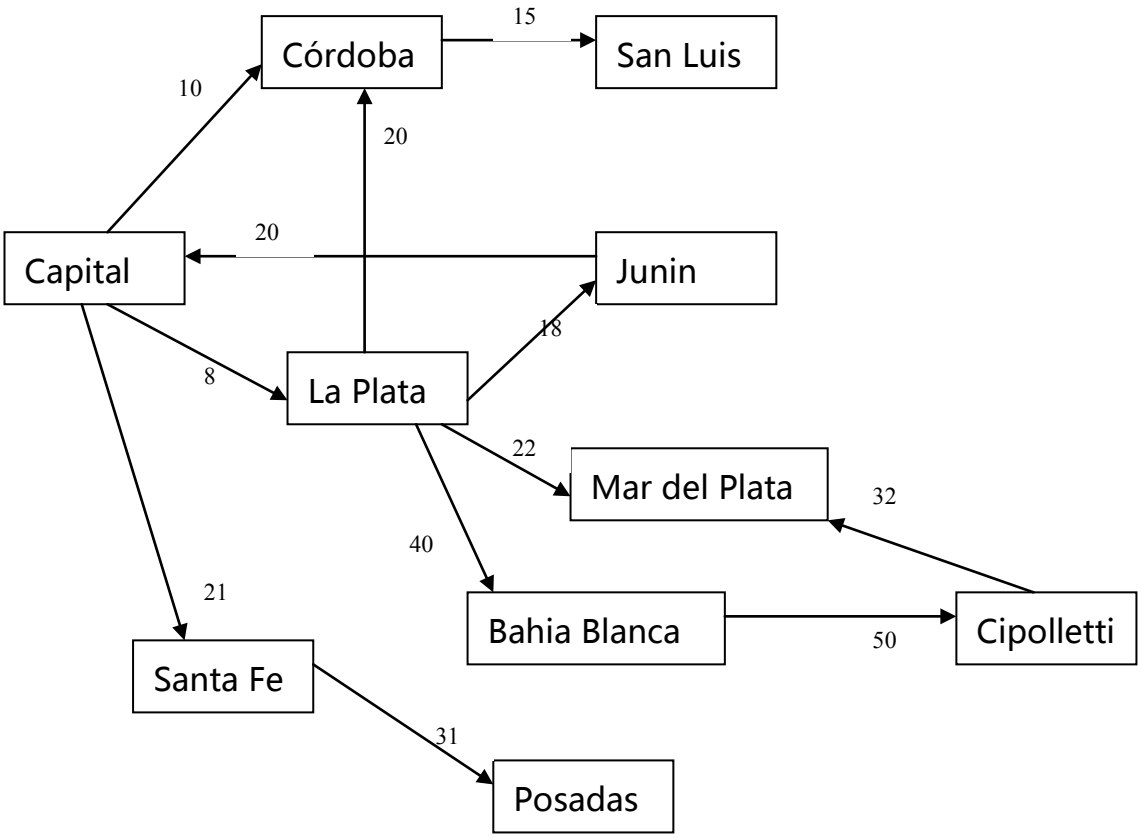
Tabla	Campos	Tipo	Tamaño
Clientes	Cli_codigo	Integer	
	Cli_nombre	String	30
	Cli_Apellido	String	30
	Cli_doc_nro	String	8
	Cli_doc_tipo	Integer	
TiposDoc	Tip_codigo	Integer	
	Tip_descrip	String	15
Domicilios	Dom_codigo	Integer	
	Cli_codigo	Integer	
	Dom_calle	String	15

	Dom_nro	String	6
	Dom_localidad	String	30
	Dom_CPA	String	10
	Dom_pcia	String	30
	Dom_telefono	String	15
	Dom_tel_internet	String	15
ServiciosInternet	Ser_codigo	Integer	
	Ser_descrip	String	255
	Ser_cant_horas	Integer	
	Ser_cant_casillas	Integer	
PreciosXServicio	Pxs_codigo	Integer	
	Ser_codigo	Integer	
	Pxs_importe	Float	
	Pxs_valor_min_exc	Float	
CabezaFacturas	Cab_codigo	Integer	
	Cab_nro	Integer	
	Cli_codigo	Integer	
	Dom_codigo	Integer	
	Cab_importe_total	Float	
ItemsFacturas	Ite_codigo	Integer	
	Cab_codigo	Integer	
	Ser_codigo	Integer	
	Ite_min_excedente	Integer	
	Ite_importe	Float	

Nodos:

1. Nodo La Plata
2. Nodo Mar del Plata
3. Nodo Capital
4. Nodo Córdoba
5. Nodo Junin
6. Nodo San Luis
7. Nodo Cipolletti
8. Nodo Santa Fe
9. Nodo Posadas
10. Nodo Bahia Blanca

Conexiones entre nodos:



Primer Etapa

Modelo 1:

- ✓ Caracterización: Preponderancia de actualizaciones (altas, bajas y modificaciones)

Corrida #1:

Características generales de la corrida:

- Preponderancia de operaciones de actualización.
- Las tablas están replicadas en un solo nodo, que a su vez es propietaria de todas. El porcentaje de replicación de tablas es del 10%.

Operaciones por Modelo:

Altas	50
Modificaciones	15
Bajas	25

Consultas	10

Operaciones por Tabla:

	Altas	Bajas	Modificaciones	Consultas
CabezaFacturas	30	0	5	7,5
Clientes	15	40	10	20
Domicilios	15	60	45	15
ItemsFacturas	30	0	5	7,5
PreciosXServicio	5	0	30	20
ServiciosInternet	5	0	5	15
TiposDoc	0	0	0	15
	100	100	100	100

Esquema de Réplicas:

Nodo	Tabla	Replicada	Propietario
La Plata	Clientes	Si	Si
	TiposDoc	Si	Si
	Domicilios	Si	Si
	ServiciosInternet	Si	Si
	PreciosXServicio	Si	Si
	CabezaFacturas	Si	Si
	ItemsFacturas	Si	Si

Resto de los nodos:

No tienen replicada *ninguna* de las tablas. Esta todo absolutamente centralizado en el nodo LA PLATA.

Corrida #2:

Características generales de la corrida:

- Preponderancia de operaciones de actualización.
- Las tablas están replicadas solo en dos nodos, siendo estos propietarios de las mismas. El porcentaje de replicación alcanza el 20%.

Operaciones por Modelo:

Idem que Corrida #1

Operaciones por Tabla:

Igual que Corrida #1

Esquema de réplicas:

Nodo	Tabla	Replicada	Propietario
Capital	Clientes	Si	Si
	TiposDoc	Si	No
	Domicilios	Si	Si
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	Si
	ItemsFacturas	Si	Si
La Plata	Clientes	Si	No
	TiposDoc	Si	Si
	Domicilios	Si	No
	ServiciosInternet	Si	Si
	PreciosXServicio	Si	Si
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No

Resto de los nodos:

No tienen replicada *ninguna* tabla. Cualquier operación debe remitirlos a los nodos master de las mismas o al nodo más cercano en caso de consultas. La centralización es en dos nodos.

Corrida #3:

Características generales de la corrida:

- Preponderancia de operaciones de actualización.
- Las tablas están replicadas solo en tres nodos, siendo sólo dos de ellos propietarios de las mismas. El porcentaje de replicación alcanza el 30%.

Operaciones por Modelo:

Igual que Corridas #1 y #2

Operaciones por Tabla:

Igual que Corridas #1 y #2

Esquema de Réplicas:

Nodo	Tabla	Replicada	Propietario

Capital	Clientes	Si	Si
	TiposDoc	Si	No
	Domicilios	Si	Si
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	Si
	ItemsFacturas	Si	Si
La Plata	Clientes	Si	No
	TiposDoc	Si	Si
	Domicilios	Si	No
	ServiciosInternet	Si	Si
	PreciosXServicio	Si	Si
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No
Córdoba	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No

Resto de los nodos:

No tienen replicada *ninguna* tabla. Cualquier operación en ellos deben remitirlas a los nodos master, o al nodo más cercano de la red en el caso de consultas.

Corrida #4:

Características generales de la corrida:

- Preponderancia de operaciones de actualización.
- Las tablas están replicadas *todas* en *todos* los nodos, pero sólo dos de ellos son propietarios de las mismas. El porcentaje de replicación es del 100%.

Operaciones por Modelo:

Igual que Corridas #1, #2 y #3

Operaciones por Tabla:

Igual que Corridas #1, #2 y #3

Esquema de Réplicas:

Nodo	Tabla	Replicada	Propietario
Capital	Clientes	Si	Si
	TiposDoc	Si	No
	Domicilios	Si	Si
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	Si
	ItemsFacturas	Si	Si
La Plata	Clientes	Si	No
	TiposDoc	Si	Si
	Domicilios	Si	No
	ServiciosInternet	Si	Si
	PreciosXServicio	Si	Si
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No

Resto de los nodos:

Tienen replicadas *todas* las tablas, pero no son propietarias de ninguna

Corrida #5:

Características generales de la corrida:

- Preponderancia de operaciones de actualización.
- Las tablas están replicadas algunos de los nodos, pero sólo dos de ellos son propietarios de las mismas. El porcentaje de replicación es del 45%.

Operaciones por Modelo:

Igual que Corridas #1, #2, #3 y #4

Operaciones por Tabla:

Igual que Corridas #1, #2, #3 y #4

Esquema de Réplicas:

Nodo	Tabla	Replicada	Propietario
Capital	Clientes	Si	Si
	TiposDoc	No	No
	Domicilios	Si	Si

	ServiciosInternet	No	No
	PreciosXServicio	No	No
	CabezaFacturas	Si	Si
	ItemsFacturas	Si	Si
La Plata	Clientes	No	No
	TiposDoc	Si	Si
	Domicilios	No	No
	ServiciosInternet	Si	Si
	PreciosXServicio	Si	Si
	CabezaFacturas	No	No
	ItemsFacturas	No	No
Córdoba	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No
Mar del Plata	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No
Junin	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	No	No
	PreciosXServicio	No	No
	CabezaFacturas	Si	No
	ItemsFacturas	No	No
Santa Fe	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No

Resto de los nodos:

No tienen *ninguna* tabla replicada. Este esquema de replicación es intermedio entre los definidos para las corridas #3 y #4.

Modelo 2:

- ✓ Caracterización: Preponderancia de consultas.

Corrida #1:

Características generales de la corrida:

- Preponderancia de operaciones de consulta.
- Las tablas están replicadas en un solo nodo, que a su vez es propietaria de todas. El porcentaje de replicación alcanza al 10%.

Operaciones por Modelo:

Altas	10
Modificaciones	10
Bajas	10
Consultas	70

Operaciones por Tabla:

	Altas	Bajas	Modificaciones	Consultas
CabezaFacturas	20	20	20	40
Clientes	20	20	20	30
Domicilios	20	20	20	10
ItemsFacturas	10	10	10	5
PreciosXServicio	10	10	10	5
ServiciosInternet	10	10	10	5
TiposDoc	10	10	10	5
	100	100	100	100

Esquema de Réplicas:

Nodo	Tabla	Replicada	Propietario
La Plata	Clientes	Si	Si
	TiposDoc	Si	Si
	Domicilios	Si	Si
	ServiciosInternet	Si	Si
	PreciosXServicio	Si	Si
	CabezaFacturas	Si	Si
	ItemsFacturas	Si	Si

Resto de los nodos:

No tienen replicada *ninguna* de las tablas. Está todo absolutamente centralizado en el nodo LA PLATA.

Corrida #2:

Características generales de la corrida:

- Preponderancia de operaciones de consulta.
- Las tablas están replicadas solo en dos nodos, siendo estos propietarios de las mismas. El porcentaje de replicación alcanza el 20%.

Operaciones por Modelo:

Igual que Corrida #1

Operaciones por Tabla:

Igual que Corrida #1

Esquema de Réplicas:

Nodo	Tabla	Replicada	Propietario
Capital	Clientes	Si	Si
	TiposDoc	Si	No
	Domicilios	Si	Si
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	Si
	ItemsFacturas	Si	Si
La Plata	Clientes	Si	No
	TiposDoc	Si	Si
	Domicilios	Si	No
	ServiciosInternet	Si	Si
	PreciosXServicio	Si	Si
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No

Resto de los nodos:

No tienen replicada *ninguna* tabla. Cualquier operación debe remitirlos a los nodos master de las mismas, o al nodo más cercano en caso de tratarse de una consulta. La centralización es en dos nodos.

Corrida #3:

Características generales de la corrida:

- Preponderancia de operaciones de consulta.
- Las tablas están replicadas solo en tres nodos, siendo sólo dos de ellos propietarios de las mismas. El porcentaje de replicación alcanza el 30%.

Operaciones por Modelo:

Igual que Corridas #1 y #2

Operaciones por Tabla:

Igual que Corridas #1 y #2

Esquema de Réplicas:

Nodo	Tabla	Replicada	Propietario

Capital	Clientes	Si	Si
	TiposDoc	Si	No
	Domicilios	Si	Si
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	Si
	ItemsFacturas	Si	Si
La Plata	Clientes	Si	No
	TiposDoc	Si	Si
	Domicilios	Si	No
	ServiciosInternet	Si	Si
	PreciosXServicio	Si	Si
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No
Córdoba	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No

Resto de los nodos:

No tienen replicada ninguna tabla. Cualquier operación en ellos deben remitirlas a los nodos master, o al nodo más cercano de la red en el caso de consultas.

Corrida #4:

Características generales de la corrida:

- Preponderancia de operaciones de consulta.
- Las tablas están replicadas *todas* en *todos* los nodos, pero sólo dos de ellos son propietarios de las mismas. El porcentaje de replicación es del 100%.

Operaciones por Modelo:

Igual que Corridas #1, #2 y #3

Operaciones por Tabla:

Igual que Corridas #1, #2 y #3

Esquema de Réplicas:

Nodo	Tabla	Replicada	Propietario
Capital	Clientes	Si	Si
	TiposDoc	Si	No
	Domicilios	Si	Si
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	Si
	ItemsFacturas	Si	Si
La Plata	Clientes	Si	No
	TiposDoc	Si	Si
	Domicilios	Si	No
	ServiciosInternet	Si	Si
	PreciosXServicio	Si	Si
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No

Resto de los nodos:

Tienen replicadas *todas* las tablas, pero no son propietarios de ninguna

Corrida #5:

Características generales de la corrida:

- Preponderancia de operaciones de consulta.
- Las tablas están replicadas *algunos* de los nodos, pero sólo dos de ellos son propietarios de las mismas. El porcentaje de replicación es del 45%.

Operaciones por Modelo:

Igual que Corridas #1, #2, #3 y #4

Operaciones por Tabla:

Igual que Corridas #1, #2, #3 y #4

Esquema de Réplicas:

Nodo	Tabla	Replicada	Propietario
Capital	Clientes	Si	Si
	TiposDoc	No	No
	Domicilios	Si	Si
	ServiciosInternet	No	No

	PreciosXServicio	No	No
	CabezaFacturas	Si	Si
	ItemsFacturas	Si	Si
La Plata	Clientes	No	No
	TiposDoc	Si	Si
	Domicilios	No	No
	ServiciosInternet	Si	Si
	PreciosXServicio	Si	Si
	CabezaFacturas	No	No
	ItemsFacturas	No	No
Córdoba	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No
Mar del Plata	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No
Junin	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	No	No
	PreciosXServicio	No	No
	CabezaFacturas	Si	No
	ItemsFacturas	No	No
Santa Fe	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No

Resto de los nodos:

No tienen *ninguna* tabla replicada. Este esquema de replicación es intermedio entre los definidos para las corridas #3 y #4.

Segunda Etapa

Modelo 3

- ✓ Caracterización: Esquema de replicación bajo, todas las tablas concentradas, y grado de replicación general es del 20%.

Corrida #1:

Características generales de la corrida:

- Preponderancia de operaciones de consultas, alcanzando estas el 70%.
- Las tablas están replicadas en dos nodos, cada uno propietario de un grupo de las mismas.

Operaciones por Modelo:

Altas	10
Modificaciones	10
Bajas	10
Consultas	70

Operaciones por Tabla:

	Altas	**Bajas**	**Modificaciones**	**Consultas**
CabezaFacturas	30	0	5	7,5
Clientes	15	40	10	20
Domicilios	15	60	45	15
ItemsFacturas	30	0	5	7,5
PreciosXServicio	5	0	30	20
ServiciosInternet	5	0	5	15
TiposDoc	0	0	0	15
	100	100	100	100

Esquema de Réplicas:

Nodo	**Tabla**	**Replicada**	**Propietario**
Capital	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	Si	Si
	PreciosXServicio	Si	Si
	CabezaFacturas	Si	Si
	ItemsFacturas	Si	Si
Mar del Plata	Clientes	Si	Si
	TiposDoc	Si	Si

	Domicilios	Si	Si
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No

Resto de los nodos:

No tienen replicadas *ninguna* de las tablas. Está todo absolutamente centralizado en el nodo CAPITAL y MAR DEL PLATA.

Corrida #2:

Características generales de la corrida:

- Preponderancia de operaciones de consulta, alcanzando el 55% de las mismas.
- Las tablas están replicadas solo en dos nodos, y siendo estos propietarios de las mismas.

Operaciones por Modelo:

Altas	15
Modificaciones	15
Bajas	15
Consultas	55

Operaciones por Tabla:

Idem corrida # 1

Esquema de Réplicas:

Idem corrida # 1

Corrida #3:

Características generales de la corrida:

- Leve preponderancia de operaciones de actualización, alcanzando éstas el 60%.

- Las tablas están replicadas solo en dos nodos, siendo estos propietarios de las mismas.

Operaciones por Modelo:

Altas	20
Modificaciones	20
Bajas	20
Consultas	40

Operaciones por Tabla:

Igual que Corridas #1 y #2

Esquema de Réplicas:

Igual que Corridas #1 y #2

Corrida #4:

Características generales de la corrida:

- Preponderancia de operaciones de actualización, alcanzando el 75%.
- Las tablas están replicadas en dos nodos, y sólo ellos son propietarios de las mismas.

Operaciones por Modelo:

Altas	25
Modificaciones	25
Bajas	25
Consultas	25

Operaciones por Tabla:

Igual que Corridas #1, #2 y #3

Esquema de Réplicas:

Igual que Corridas #1, #2 y #3

Corrida #5:

Características generales de la corrida:

- Preponderancia de operaciones de actualización, alcanzando el 90%.

- Las tablas están replicadas en dos de los nodos, pero sólo dos son propietarios de las mismas.

Operaciones por Modelo:

Altas	30
Modificaciones	30
Bajas	30
Consultas	10

Operaciones por Tabla:

Igual que Corridas #1, #2, #3 y #4

Esquema de Réplicas:

Igual que Corridas #1, #2, #3 y #4

Modelo 4

- ✓ Caracterización: Alto nivel de replicación, todos los nodos poseen réplicas de las todas o algunas de las tablas. El grado de replicación es del 100%

Corrida #1:

Características generales de la corrida:

- Preponderancia de operaciones de consulta, que alcanzan el 70%.
- Las tablas están replicadas en todos los nodos.

Operaciones por Modelo:

Altas	10
Modificaciones	10
Bajas	10
Consultas	70

Operaciones por Tabla:

	Altas	**Bajas**	**Modificaciones**	**Consultas**
CabezaFacturas	30	0	5	7,5
Clientes	15	40	10	20
Domicilios	15	60	45	15
ItemsFacturas	30	0	5	7,5
PreciosXServicio	5	0	30	20

ServiciosInternet	5	0	5	15
TiposDoc	0	0	0	15
	100	100	100	100

Esquema de Réplicas:

Nodo	Tabla	Replicada	Propietario
Capital	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	Si	Si
	PreciosXServicio	Si	Si
	CabezaFacturas	Si	Si
	ItemsFacturas	Si	Si
Córdoba	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No
La Plata	Clientes	Si	Si
	TiposDoc	Si	Si
	Domicilios	Si	Si
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No
Santa Fe	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No
San Luis	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No
Junin	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No

	CabezaFacturas	Si	No
	ItemsFacturas	Si	No
Mar del Plata	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No
Bahia Blanca	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No
Cipolletti	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No
Posadas	Clientes	Si	No
	TiposDoc	Si	No
	Domicilios	Si	No
	ServiciosInternet	Si	No
	PreciosXServicio	Si	No
	CabezaFacturas	Si	No
	ItemsFacturas	Si	No

Corrida #2:

Características generales de la corrida:

- Leve preponderancia de operaciones de consulta, siendo estas del 55%.
- Las tablas están replicadas solo en dos nodos, y estos son propietarios de las mismas.

Operaciones por Modelo:

Altas	15
Modificaciones	15
Bajas	15
Consultas	55

Operaciones por Tabla:

Igual que Corrida #1

Esquema de Réplicas:

Igual que Corrida #1

Corrida #3:

Características generales de la corrida:

- Leve preponderancia de operaciones de actualización, alcanzando el 60% de ellas.
- Las tablas están replicadas en todos los nodos, sólo dos son propietarios de las mismas.

Operaciones por Modelo:

Altas	20
Modificaciones	20
Bajas	20
Consultas	40

Operaciones por Tabla:

Igual que Corridas #1 y #2

Esquema de Réplicas:

Igual que Corridas #1 y #2

Corrida #4:

Características generales de la corrida:

- Preponderancia de operaciones de actualización, que alcanzan el 75%.
- Las tablas están replicadas todas en todos los nodos, pero sólo dos son propietarios de las mismas.

Operaciones por Modelo:

Altas	25
Modificaciones	25

Bajas	25
Consultas	25

Operaciones por Tabla:

Igual que Corridas #1, #2 y #3

Esquema de Réplicas:

Igual que Corridas #1, #2 y #3

Corrida #5:

Características generales de la corrida:

- Preponderancia de operaciones de actualización, que alcanzan el 90%.
- Las tablas están replicadas en todos los nodos, pero sólo dos son propietarios de las mismas.

Operaciones por Modelo:

Altas	30
Modificaciones	30
Bajas	30
Consultas	10

Operaciones por Tabla:

Igual que Corridas #1, #2, #3 y #4

Esquema de Réplicas:

Igual que Corridas #1, #2, #3 y #4

Bibliografía

[ABKW96] "Replication, Consistency and Practicality: Are these mutually exclusive?" Todd Anderson, Yuri Breitbart , Henry F. Korth, Avishai Wool. ACM 1996.

[Atr93] "Distributed Databases". Atre. McGraw-Hill, 1993

[BKRSS99] "Update Propagation Protocols for Replicated Databases". Yuri Breitbart, Raghavan Komondoor, Rajeev Rastogi, S. Seshadri, Avi Silverschatz. SIGMOD 99.

[BOB96] "Distributed and Multidatabase Systems". Angelo R. Bobak. 1996, Artech House, Inc.

[Bru95] "Experimental Evaluation of Dynamic Data Allocation Strategies in a Distributed Database With Changing Workloads". Brunston A., Leutenegger S., Simha R. ACM, 1995

[BUR94] "Managing Distributed Databases. Building bridges between database islands". Donald K. Burleson. 1994. John Wiley & Sons, Inc.

[Cam96] "The Java Tutorial". Mary Campione, K. Walrath, Addison-Wesley, 1996.

[DBDZ85] "Replicating and Allocating Data in a Distributed Database System for Workstations". Andreas R. Diener, Richard P. Bragger, Andreas Dudier, Carl A. Zehnder. ACM 1985.

[Eck97] "Thinking in Java". B. Eckel. MindView Inc., 1997

[GHOS96] "The dangers of replication and a solution". Jim Gray, Pat Helland, Patrick O'Neil, Dennis Shasha. AMC 1996.

[Gol95] "Things every update replication customer should know". Rob Goldring. ACM. 1995.

[HEH98] "Enabling Large-scale Simulations: Selective Abstraction Approach to the Study of Multicast Protocols". Polly Huang, Deborah Estrin, John Heidermann. International Symposium on Modeling. Montreal, Canada. Julio 1998.

[HER87] "Concurrency versus Availability: Atomicity Mechanisms for Replicated Data". Maurice Herlihy - ACM Transactions of Computer Systems. 1987

[JAV01] "The Java Tutorial, a practical guide for programmers". www.java.sun.com.

[JAV02] "Java desde cero". www.cybercursos.net.

[KeiChi] A new replication strategy for unforeseeable disconnection under Agent-Based Mobile Computing Systems". Keith Lee, Y.H.Chin, 1996.

[KHK98] "The future of Java-Based simulation". Richard A. Kilgore, Kevin J. Healy, George B. Kleindorfer. Proceedings of the 1998 Winter Simulation Conference.

[Len01] "Bases de datos Distribuídas. Estudio de Actualización de Réplicas de datos". Sergio Len. Trabajo de Grado,Lic. en Informática. UNLP. 2001.

[MR95] "Allocating Data and Operations to Nodes in Distributed Database Design". Salvatore T. March, Sangkyu Rho. IEEE. 1995

[OAW97] " Java Threads". Oaks S. and H. Wong. 1997. O´Reilly & Associates, Inc

[OVA91] "Principles of Distributed Database Systems". M. Tamer Özsu, Patrick Valduriez. 1991. Prentice Hall.

[ÖZV96] "Distributed and Parallel Database Systems". M. Tamer Özsu, Patrick Valduriez. ACM Computing Surveys, Vol. 28, No. 1, Marzo de 1996.

[PSM98] "Improving Data Freshness in Lazy Master Schemes". Esther Pacitti, Eric Simon, Rubens Melo. IEEE. Mayo 1998.

[PUL91] "Replica Control in Distributed Systems: An Asynchronous Approach". Calton Pu, Avraham Leff. ACM 1991.

[SAH96] "An Analysis of the Average Message Overhead in Replica Control Protocols". Debanjan Saha, Sampath Rangarajan and Satish K. Tripathi. IEEE, Transactions on parallel and Distributed Systems, Vol 7 No. 10, October 1996.

[SK91] "Fundamentos de Bases de Datos". Abraham Silberschatz, Henry F. Korth S. Sudarshan. Tercera edición. Ed Mc Graw Hill. 1991.

[SPL92] "A Simulation Study of Replication Control Protocols Using Volatile Witnesses". Perry K. Sloope, Jehan-Francois Pâris, Daniel D.E.Long. IEEE 1992

[Sun96] "JDBC: a Java Sql API". Sun Microsystems Inc, 1997

[WOJ92] "Distributed algorithms for dynamic replication of data". Ouri Wolfson, Sushil Jajodia. ACM. 1992.

[ZHHO] "The Design and Simulation of a Hybrid Replication Control Protocol". Wanlei Zhou, Robert Holmes. School of Computing and Mathematics. Dehain University. Geelong, VIC3217, Australia.

Tabla de Contenidos