Experiencias en el análisis de fallas en Bases de Datos distribuidas.

Ivana Miatón[1]
Sebastián Ruscuni[1]
Lic. Rodolfo Bertone [2]
Ing. A.De Giusti[3]

Laboratorio de Investigación y Desarrollo en Informática[4]
Departamento de Informática - Facultad de Ciencias Exactas
Universidad Nacional de La Plata

Resumen

Se presenta un trabajo experimental de evaluación y recuperación de fallas en Bases de Datos distribuidas, utilizando un modelo de arquitectura distribuida, conformado por una red local de procesadores heterogéneos vinculados mediante PVM.

Se analizan las fallas que pueden producirse en una transacción global, utilizando el protocolo de commit de dos fases, modelizando y simulando situaciones de caída de alguno de los sitios de la red o de falla en el vínculo de comunicaciones (total o parcial).

Se presentan resultados y conclusiones relacionados con el modelo experimentado y se señalan las líneas de investigación futura en el tema.

Palabras clave: *Bases de Datos distribuidas. Recuperación de fallas. Procesamiento distribuido.*

[1] Alumnos avanzados de la Licenciatura en Informática. Departamento de Informática, Facultad de Ciencias Exactas, UNLP. Becarios Ad-honorem L.I.D.I.
E-mail imiaton,sruscuni@lidi.info.unlp.edu.ar
[2] Prof. Adjunto con Dedicación Exclusiva, LIDI. Departamento de Informática, Facultad de Ciencias Exactas, UNLP.
E-mail pbertone@lidi.info.unlp.edu.ar
[3] Inv. Principal CONICET. Profesor Titular Dedicación Exclusiva, Departamento de Informática, Facultad de Ciencias. Exactas, UNLP.
E-mail degiusti@lidi.info.unlp.edu.ar
[4] Calle 50 y 115 Primer Piso, (1900) La Plata, Argentina, Teléfono +54 21 22 7707
WEB: lidi.info.unlp.edu.ar

Introducción

Se define un sistema distribuido como una colección de computadoras autónomas interconectadas mediante una red, con un software diseñado para brindar facilidades integradas de cómputo. [COUL95]. Estos sistemas distribuidos están implementados sobre plataformas de hardware que varían en su tamaño y conexión.

Las características claves de un sistema distribuido son: soportes para compartir recursos, concurrencia, escalabilidad, tolerancia a fallos y transparencia.

El rango de aplicaciones distribuidas es por demás amplio, desde la provisión de facilidades de cómputo para grupos de usuarios hasta sistemas de comunicación multimedia o bancarios automatizados.

La historia de los sistemas distribuidos se origina en el desarrollo de computadoras multiusuario y redes de computadoras en los años sesenta, modelos que fueron estimulados en la década del 70 con la baja notable en los costos de las computadoras personales, el surgimiento de redes locales y el sistema operativo UNIX.

De la misma forma en que el hardware y los soportes de redes fueron evolucionando, se observó el mismo cambio o evolución en los sistemas de bases de datos. En un principio las bases de datos fueron monousuarias, luego esta evolución llevó a bases de datos centralizadas, donde los usuarios acceden a los dados ubicados en una única computadora, puede ocurrir que este usuario esté interconectado a través de una red de área local, o eventualmente que su localidad esté distribuida dentro de una red de área global. La siguiente evolución en bases de datos llevó a distribuir la información, los datos dejaron de estar ubicados en una computadora y se distribuyeron geográficamente en varias workstation distribuidas en una WAN. [KROE96]

Hay varias razones para desarrollar y usar un sistema de bases de datos distribuidas (BDD): [HANS97]

- ✓ Las organizaciones tienen divisiones en diferentes localidades. Para cada localidad puede haber un conjunto de datos que usan frecuente y exclusivamente.

- ✓ Permitir que cada sitio almacene y mantenga su propia base de datos facilita el acceso inmediato y eficaz a los datos que se usan con más frecuencia.

- ✓ Las bases de datos distribuidas logran mejoras en la fiabilidad del sistema, aunque agregan otros elementos que hacen al mantenimiento de seguridad e integridad de información.

- ✓ Permitir control local de los datos que se usan con mayor frecuencia en un sitio, logrando un mejor grado de satisfacción de los usuarios.

Con el advenimiento de nuevo hardware (workstation más poderosas y de menor precio) y del software de redes robusto y confiable, el uso de grandes computadoras se vio limitado para migrar a redes locales o globales. Asociado a esto,

aparece un concepto nuevo, *downsizing*, que se refiere a migración de aplicaciones existentes para mainframes a redes de workstations. [UMAR93]

Para efectuar el downsize de una aplicación, el diseñador debe elegir migrar la aplicación completa a una estación de trabajo, o dividir manejo de datos, interfaces de usuario y lógica de la aplicación entre diferentes computadoras, utilizando por ejemplo un modelo Client Server. [BERS92]. Cuando se analiza la división de los datos, teniendo en cuenta las características disponibles con BDD, es posible que la migración de los mismos incluya distribuirlos en varias computadoras de la red.

Este trabajo, desarrollado básicamente por alumnos que se inician en la investigación en el LIDI, se analizan los problemas asociados con la distribución de información a lo largo de redes de computadoras heterogéneas, y se evalúan cuantitativamente las principales soluciones de los problemas clásicos existentes.

Algunos conceptos de Bases de Datos Distribuidas.

Un sistema de bases de datos distribuidas (SBDD) está formado por una colección de sitios, cada uno de los cuales opera un sistema de BD para el procesamiento de las actividades que sólo requieren de datos locales. Adicionalmente cada lugar puede procesar transacciones que requieren datos que están almacenados en otros sitios.

Entonces es posible administrar transacciones locales y globales. Las primeras involucran transacciones que se generan en una localidad y utilizan datos ubicados en la misma. En tanto, cuando se utilizan datos de otras localidades, la transacción se convierte en global.

El diseño de BDD puede resultar en una tarea compleja. Se deben hacer consideraciones cuidadosas sobre los objetivos y estrategias, las cuales pueden resumirse como: [DATE87]

- ✓ *Transparencia de la ubicación*: permite al usuario acceder a los datos sin conocer el lugar donde los mismos residen.
- ✓ *Transparencia de la duplicación*: si existe más de una copia de los datos, solo se debe escoger una para trabajar, pero todas las copias deben ser actualizadas.
- ✓ *Independencia de la configuración*: permite a la organización añadir o reemplazar hardware sin tener que estar cambiando componentes de software existentes.
- ✓ *DDBMS heterogéneos*: disponer a lo largo de la organización de diversos DBMS y que los mismos interactúen en conjunto en forma transparente.
- ✓ *Duplicación de datos*: en un entorno distribuido la duplicación de datos aumenta la disponibilidad del sistema; cuando una copia no está disponible por un fallo de un sitio, se puede acceder a una copia que se encuentra en otra localidad

- ✓ *Fragmentación de datos*: se refiere a la forma en que las relaciones disponibles en una base de datos se pueden subdividir y distribuir entre los sitios de la red. Esta subdivisión puede efectuarse de dos maneras:
  - ✓ *Horizontal*: se generan varios conjuntos de tuplas a partir de una relación y los mismos son almacenados en distintas localidades.
  - ✓ *Vertical*: los atributos de una relación son subdivididos y almacenados en diversas estaciones de trabajo.

Desde el punto de vista de seguridad e integridad de los datos, una BDD debe proveer los mismos conceptos que los sistemas centralizados:

- ✓ Las transacciones deben conservar el concepto de atomicidad, o sea deben ejecutarse por completo o no hacerse.
- ✓ Los fallos de un sistema centralizado se mantienen agregando, ahora, problemas en las comunicaciones, caídas de nodos o localidades, pérdida de mensajes o fragmentación de la red.

Los protocolos clásicos de recuperación para mantener la integridad de los datos son: (1) el protocolo de dos fases, (2) protocolo de tres fases.

Al igual que en los sistemas centralizados, debe proveerse una política para el control de concurrencia. La diferencia fundamental de un sistema de datos distribuidos radica en la posibilidad de contar con un mecanismo que tenga en cuenta, y resuelva, los problemas de replicación de datos. [BERN83] [ELBA89]

El procesamiento de consultas, por último, es más eficiente. El gestor de base de datos local de cada nodo tiene la posibilidad de analizar la consulta y subdividirla en diferentes partes, cada una de las cuales se ejecutará sobre diferentes localidades de la red. Luego recepciona las respuestas y confecciona la salida hacia el usuario. De esta manera se logra un mayor paralelismo en el procesamiento de una solución. La frase *nada es gratis* es aplicable en este punto; si bien se obtiene mayor paralelismo en el procesamiento de la respuesta hay un costo adicional consistente en la transmisión de los datos que forman la consulta.

Protocolo de dos fases

Cada transacción se genera en una localidad, el coordinador local de transacciones es el encargado de controlar la ejecución de la misma. En caso de ser una transacción local, se procede igual a los sistemas centralizados. Ahora, si la transacción accede a datos globales, el coordinador debe encargarse de subdividir la transacción en subtransacciones que se ejecutarán en las localidades correspondientes. Cada localidad participante recibe la subtransacción y la procesa como si esta fuera una local. [WEB1] [WEB2]

El coordinador debe, una vez que el procesamiento finaliza decidir en que estado termina la transacción correspondiente. [WEB3] Para ello activa el protocolo de dos fases:

- ✓ *Fase 1:* el coordinador envía un mensaje a cada localidad involucrada preguntando si se pudo finalizar la transacción. Ante una respuesta negativa, o eventualmente, falta de respuesta, la transacción debe ser abortada. Si todos contestan afirmativamente la transacción está en condiciones de ser cometida.

- ✓ *Fase 2*: se envía un mensaje de abortar (si hay al menos una respuesta negativa o se cumple un tiempo de espera predeterminado), o un mensaje de finalizar (si todas incluyendo al coordinador pudieron finalizar la transacción. Cada localidad comete la transacción y envía un mensaje de reconocimiento al coordinador.

El protocolo que se utiliza incorpora un *timeout,* es decir, se tiene en cuenta el tiempo de espera máximo entre nodos. Si una subtransacción no recibe un mensaje del coordinador le pide a todas las participantes un mensaje de ayuda. Cuando alguna recibe este mensaje puede ayudar a su compañera, de acuerdo a su propio estado; es decir, si ha recibido del coordinador un mensaje commit o abort, envía el mismo. La transacción en espera puede ahora realizar el commit o abortar sin ningún lugar a dudas, pues su compañera ha replicado el mensaje seguro del coordinador. Por el contrario, si la subtransacción no recibe ningún mensaje, continuará en espera. [ZANC96] [BERN84]

El número de mensaje si se tienen M localidades involucradas está acotado a 4 * (M -1), siendo este un valor demasiado alto para eventualmente no llegar a cometer la transacción. El protocolo de tres fases presenta una mejor política para el tratamiento de mensajes pero por su concepción presenta mayores inconvenientes al momento de implementación, por lo tanto es generalmente descartado como posible solución. [DATE90] [KORT94]

Modelo de Arquitectura distribuida de soporte para experimentación

Se cuenta con una red de procesadores heterogéneos que está formado por un gran número de computadoras o nodos linkeados por una red de interconexión. En ellos se pueden observar diferentes tipos de heterogeneidad con respecto a :

- ✓ arquitectura

- ✓ formato de datos,

- ✓ velocidad de procesamiento,

- ✓ capacidad de almacenamiento.

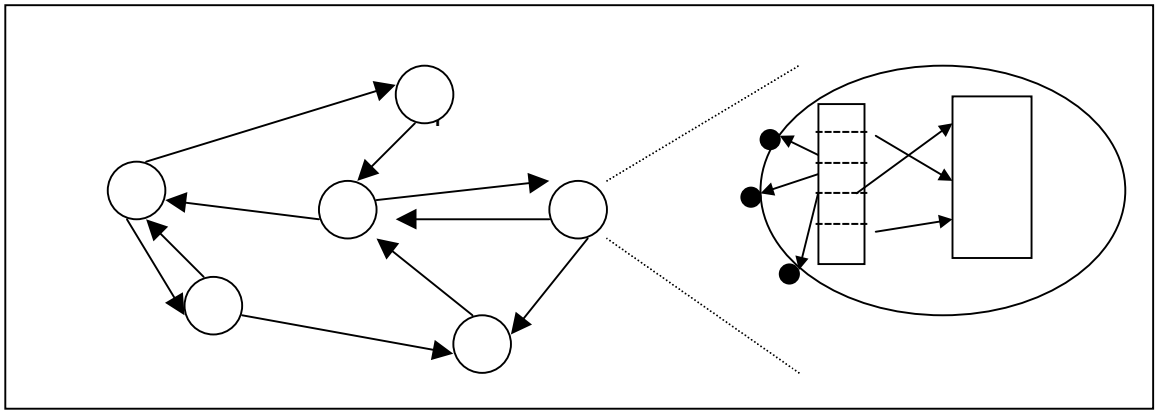
Cada procesador ejecuta su propio programa el cual puede acceder a memoria local o enviar y recibir mensajes de la red. Dichos mensajes son utilizados para comunicarse con otras computadoras o, equivalentemente, para leer y escribir en memorias remotas.

PVM (Parallel Virtual Machine) es un paquete de software que provee protocolos para la comunicación entre computadoras heterogéneas conectadas a través

de una red, haciendo que ésta se vea como una única máquina virtual paralela. Su objetivo principal es permitir que un conjunto de computadoras sean usadas cooperativamente para procesamiento concurrente o paralelo. [MORS94] [WEB4]

Utiliza el modelo de pasaje de mensajes. Maneja transparentemente todos los mensajes de ruteo, conversión de datos y manipulación de tareas que se puedan hallar en una red compuesta por computadoras de arquitecturas heterogéneas, produciendo un software altamente portable. [GEIS94]

La unidad de paralelismo es la *tarea*: un proceso secuencial independiente de control que alterna entre comunicación y procesamiento. La división de tareas se basa en un paralelismo funcional (cada tarea realiza una función diferente), o en un paralelismo de datos (varias tareas realizan la misma función pero cada una sólo conoce y procesa una parte de los mismos). [HWAN93]



La Figura anterior muestra un modelo simple de programación paralela. Está formado por un conjunto de tareas (representadas por círculos) que están conectadas por canales (indicadas con flechas). Una tarea encapsula un programa y su memoria local y además define una serie de puertos que representan la interface con su ambiente. Un canal es una cola de mensajes en donde el emisor lo utiliza para depositar nuevos y el receptor, si hay alguno disponible, los elimina.

Aquí se presenta una simulación, bajo una red heterogénea con soporte PVM de fallos que pueden producirse durante la ejecución de una transacción global utilizando el protocolo de Commit de dos fases. Se analiza principalmente la recuperación del sistema ante caídas de alguna de las localidades participantes o del entorno de red en la ejecución de una transacción distribuida.

Se cuenta con una base de datos distribuida *fragmentada horizontalmente*, es decir, para cualquier relación de la base de datos, no necesariamente se almacenará completamente en un único sitio. Los subconjuntos se distribuyen entre varios lugares por consideraciones de rendimiento. Además, en este análisis inicial no se consideraron casos de replicación de datos. [SCHU97]

Se crearon distintos procesos, aprovechando el potencial de PVM, con el objetivo de representar al sistema distribuido. Existe un proceso maestro que simula al

coordinador y una serie de procesos esclavos que serán las *localidades participantes* de la transacción en cuestión.

Principalmente utilizamos el pasaje de mensajes de PVM para representar el diálogo que existe entre el coordinador y las localidades que se llevan a cabo tanto en la Fase de Preparar (Fase 1) como en la Fase de Commit (Fase 2).

Especificación de los casos de falla a estudiar. Implementación.

Las distintas *situaciones de fallo* que pueden ocurrir durante la ejecución de una transacción distribuida son:

- ✓ *Fallo de una localidad participante*. Para recuperarse, debe examinar su bitácora y a partir de allí, decidir el destino de la transacción.

- ✓ *Fallo del coordinador*. En este caso, las localidades participantes son quienes deben decidir el destino de la transacción. Si no poseen suficiente información, tendrán que esperar a que se recupere el coordinador.

Para la implementación de una solución al problema, hay que tener en cuenta que las acciones de todas las localidades y del coordinador se ven reflejados en un archivo "Bitácora" que contiene registros donde se indica el tipo de operación (comienzo, cometido, abortado, reconocimiento de escritura). Además, ante una instrucción de modificación de datos se opera con la técnica de modificación inmediata de los datos, debiendo almacenar el valor viejo y el valor nuevo.

La secuencia de tareas que realiza el proceso maestro (*coordinador*) son:

- ✓ Dispara una tarea por cada localidad donde se realiza la transacción.
- ✓ Envía a cada localidad indicando que debe hacer
- ✓ Envía luego un mensaje de "Preparar"
- ✓ Espera una respuesta:
  - ✓ si no recibe ninguna en un tiempo determinado decide abortar
  - ✓ si recibe todos mensajes "Lista" decide ejecutar. Con al menos recibir un "Abortar", determinara que aborte la transacción.
- ✓ Envía un mensaje a todas las localidades informando lo que decidió
- ✓ Se queda esperando los reconocimientos de todos los participantes
- ✓ Elimina a las tareas que le ayudaron a realizar la transacción
- ✓ Termina

Por otro lado, las tareas que realizan cada uno de los procesos "Localidad" son:

- ✓ Recibe del coordinador/maestro las indicaciones sobre la tarea a realizar.
- ✓ Realiza las tareas indicadas.
- ✓ Espera del coordinador un mensaje "Preparar"
- ✓ Chequea si está dispuesto a realizar la transacción
- ✓ Si lo está envía al coordinador un mensaje "Lista", de lo contrario, le envía un "Abort" y ejecuta un *undo* (para dejar la base de datos consistente, eliminando las modificaciones realizadas).
- ✓ Se queda esperando respuesta del coordinador, que puede ser "Commit " o "Abort"
- ✓ Si recibió un "Abort" realiza un *undo*
- ✓ Se queda esperando en un loop esperando mensaje de las localidades hermanas. Cuando recibe uno, le informa del estado actual (Commit o Abort)
- ✓ Estas localidades terminan cuando lo decide el coordinador.

Casos de fallos tratados

Fallo de una Localidad:

Manejamos los fallos que pueden ocurrir en cuatro lugares claves durante el procesamiento. En todos los casos, se examina el archivo Bitácora y se decide qué hacer:

- ✓ *Antes de recibir el "Preparar"*. En este caso se asume que el coordinador, como no va a recibir respuesta de ella, decidió abortar, por lo tanto, esta localidad decide abortar.
- ✓ *Luego de recibir el "Preparar"* y antes de enviar el *"Lista"*. Aquí sucede algo muy similar al caso anterior, por lo que también se decide abortar.
- ✓ *Luego de haber enviado el "Lista" y antes de recibir alguna respuesta del coordinador*. En este caso la localidad tendrá que comunicarse con el coordinador (o eventualmente con alguna de las localidades hermanas por si el coordinador también falle) para averiguar cual fue el destino de la transacción.
- ✓ *Luego de hacer "Commit" o "Abort".* Aquí se sabe el destino de la transacción y lo que se hace solamente es enviar el *"reconocimiento"* al coordinador.

Fallo del coordinador:

Si el coordinador falla en la mitad de la ejecución de la transacción entonces las localidades participantes deben decidir el destino de la misma. Existirán casos en

donde las localidades no podrán determinar si ejecutar y abortar la transacción, y, por lo tanto, será necesario que esperen a que se recupere el coordinador que falló.

En todos los casos, luego que la localidad se recupera, le envía el *"reconocimiento"* al coordinador, estableciendo así el fin de la transacción. En el momento en que el coordinador recibe el reconocimiento de todas las localidades participantes culmina la simulación eliminando las tareas actuantes.

Resultados experimentales obtenidos.

Se realizaron trazas de simulación teniendo en cuenta las situaciones de fallas anteriormente explicadas. Las soluciones implementadas fueron las clásicas previstas para los problemas en cuestión, siendo el objetivo principal medir tiempos, en unidades relativas, focalizándonos principalmente en los procesos de recuperación del sistema, con el fin de mantener la *consistencia* de la base de datos.

La metodología empleada para generar los casos de prueba, reflejó los problemas reales de entornos distribuidos, como ser fallo de las comunicaciones entre localidades, fallo total o parcial de un localidad, fallo de la localidad coordinadora de la transacción activa. Se trató de evaluar, además, situaciones de fragmentación de la red producida por la falla de la comunicación o por la caída de alguna localidad que actuara como nexo.

La primeras dos fallas descriptas, en donde la localidad se cae antes de recibir el *"Preparar"* del coordinador, o luego de recibirlo y antes de enviar el *"Lista"*, son muy similares. Si damos por descontado que el tiempo en que la localidad está fallando es mayor al timeout del coordinador, entonces podemos decir que afectará considerablemente al tiempo final ya que el coordinador espera una respuesta de la localidad (que nunca llegará) para luego decidir el aborto de la transacción. También se debe tener en cuenta que en estos dos casos se realiza el *undo* ya que podemos afirmar que el destino de la transacción será abortar.

Si el fallo sucede luego de haber enviado el *"Lista"* y antes de recibir alguna respuesta del coordinador la primera fase del protocolo se realiza sin problemas, luego el coordinador envía el *commit* o *abort* sin saber que una localidad ha fallado, por lo que el tiempo tampoco se ve afectado. La recuperación de localidad consistirá en comunicarse con el coordinador para averiguar el destino de la transacción y en el caso de que se deba abortar se tendría que realizar el *undo*. Entonces el tiempo final de procesamiento podría ser menor a los anteriores.

Y por último, si falla luego de hacer *commit* o *abort*, solo afectará al tiempo la espera que realiza el coordinador del reconocimiento de la localidad que falló, ya que el resto del proceso se hace sin inconvenientes, porque luego de la caída, la localidad puede deducir con la ayuda de su bitácora el destino de la transacción. En este caso también dependerá de la decisión del coordinador si necesita realizar el *undo* o no.

El trabajo realizado se centró en simular el comportamiento de una base de datos distribuida manteniendo la integridad de la misma. Se contó con datos distribuidos pero sin la existencia de replicación ni fragmentación. Con esta etapa cubierta nos

abocamos ahora a replicar la información en distintos servers estudiando el comportamiento de las consultas y las recuperaciones del sistema en casos de fallo.

Conclusiones. Líneas de trabajo.

Disponer de un ambiente de prueba para problemas de BDD fue la principal motivación para empezar con esta línea de investigación. Como segundo objetivo se tuvo en cuenta el análisis y generación de casos de prueba o trazas para evaluar el comportamiento del prototipo, ante las diversas situaciones presentadas.

Como se planteó anteriormente, el protocolo de integridad fue el dos fases, debido a las ventajas que presenta para su implementación sobre el protocolo de tres fases. La decisión respecto de la modificación inmediata de la base de datos fue solamente por cuestiones de implementación.

Los trabajos que se están realizando actualmente están relacionados con incorporar al análisis casos de replicación de información. Esto conlleva el estudio del manejo de concurrencia en el acceso a los datos, para lo cual también se utiliza el protocolo de dos fases.

El objetivo que se pretende alcanzar es el estudio estadístico de performance con y sin replicación, evaluando la disponibilidad de información, aumentando el paralelismo, versus la degradación de performance ante actualizaciones o recuperación de errores.

Si bien las técnicas de pruebas realizadas en las experiencias presentadas en este artículo fueron clásicas, la estructuración de cada uno de los casos de prueba, nos permitieron establecer un conjunto de pautas con las cuales estamos implementando los casos de evaluación al incorporar, en la simulación, replicación de información. Una vez finalizados los estudio previstos, es nuestra intención discutir las conclusiones alcanzadas.

En estos temas se colabora con el grupo de la Universidad Nacional del Sur que dirige el profesor Ardenghi.

Además, como trabajo futuro se prevé el estudio de casos especiales, como por ejemplo en sistemas de tiempo real duro, donde agregaremos restricciones rígidas de tiempo a nuestro problema.

Bibliografía

[BERN83] *The Failure and Recovery Problem for Replicated Databasees* Bernstein, Goodman. Proc. Second ACM Symp. Principles of Distributed Computing, pp. 114-122. Aug. 1983.

[BERN84] *An algorithm for concurrency Control and Recovery in Replicated Distributed Databases* Bernstein, Goodman. ACM Trans. Database Systems, vol 9 no. 4, pp. 596-615, Dec 1984.

[BERS92] *Client Server Architecture*, Alex Berson. Mc Graw Hill Series on Computer Communication 1992.

[COUL95] *Distributed Systems, Concepts and Design*, George Coulouris, Jean Dollimore, Tim Kindberg. Addison Wesley 1995.

[DATE87] *Info Data Bases* magazine, C. J. Date, Summer 1987

[DATE90] *Introducción a los sistemas de Bases de Datos,* C. J. Date Addison Wesley, 1990.

[ELAB89] *Avaibility in Partitioned Replicated Databases* El Abbadi, Toueg, ACM Trans. Databases Systems, vol 14 no.2 pp. 264-290, June 1989.

[GEIS94] *PVM: Paralell Virtual Machine. A user guide and tutorial for networked parallel computer.* Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, Vaidy Sunderam, The MIT Press, 1994.

[HANS97] *Diseño y Administración de Bases de Datos.* Gary Hansen, Janes Hansen. Prentice Hall 1997.

[HWAN93] *Advanced Computer Architecture: Paralelism, Scalability, Programability*, Kai Hwang McGraw-Hill, 1993.

[KORT94] *Fundamentos de Bases de Datos*, Korth, Silberchatz, Mc Graw Hill, 1994.

[KRON96] *Procesamiento de bases de datos. Fundamentos, diseño e instrumentos,* David Kroenke. Prentice Hall 1996.

[MORS94] *Practical Parallel Computing*, Stephen Morse, AP Profesional 1994.

[SCHU97] *Replication, the next generation of distributed database technology*, George Shussel, www.dciexpo.com/geos/replica.html

[UMAR93] *Distributes computign and Client Server Systems*, Amjad Umar. Prentice Hall 1993.

[WEB1] *www.seas.gwu.edu/faculty/shmuel/cs267/textbook/tpcp.html*

[WEB2] *www.sei.cmu.edu/str/descriptions/dtpc_body.html*

[WEB3] *www.www.datanetbbs.com.br/dclobato/textos/bddcs/parte2.html*

[WEB4] *www.mcs.anl.gov/dbpp/text/book.html*

[ZANC96] *Análisis de Replicación en Bases de Datos Distribuidas*, Marcelo Zanconi, Tesis de Magister en Ciencias de la Computación, Univ. Nac. Del Sur, Bahía Blanca, 1996