“Desarrollo e Implementación de un Sistema de Cómputo Distribuido. Aplicación al Manejo y Operación de Bases de Datos”

TESIS

Que para obtener el grado de
MAESTRA EN CIENCIAS EN INGENIERÍA ELÉCTRICA

Presenta
VIOLETA MEDINA RIOS

DR. J. AURELIO MEDINA RIOS
Director de Tesis

DR. JUAN MANUEL GARCÍA GARCÍA
Co-Director de Tesis

UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

Agosto de 2008

DESARROLLO E IMPLEMENTACIÓN DE UN SISTEMA DE CÓMPUTO DISTRIBUIDO. APLICACIÓN AL MANEJO Y OPERACIÓN DE BASES DE DATOS

Los miembros del Jurado de Examen de Grado aprueban
la **Tesis** de **Maestría en Ciencias en Ingeniería Eléctrica** de *Violeta Medina Rios*

Dr. Juan José Flores Romero
Presidente del Jurado

Dr. J. Aurelio Medina Rios
Director de Tesis

Dr. Juan Manuel García García
Co-Director de Tesis

Dr. Antonio Ramos Paz
Vocal

Dr. Domingo Torres Lucio
Examinador Externo
Instituto Tecnológico de Morelia,
Morelia, Michoacán

Dr. J. Aurelio Medina Rios
Jefe de la División de Estudios de Posgrado
de la Facultad de Ingeniería Eléctrica, UMSNH

UNIVERSIDAD MICHOACANA DE SAN NICOLAS DE HIDALGO
Agosto del 2008

Agradecimientos

A mis padres, que en todo momento de la vida han sido un soporte incondicional. Gracias mamá, por las bendiciones que recibo de ti cada día. Gracias papá porque me alentaste a iniciar con este proyecto y porque sé que desde cualquier lugar en el que te encuentres, siempre has estado a mi lado.

Gracias a mis hermanos por el cariño y comprensión que en todo momento me han brindado, por el apoyo que ha recibido de su parte para concluir esta meta profesional.
Gracias Pepe porque has sido mi guía.
Gracias Hortensia por estar siempre a mi lado, por tu palabra de aliento, por escucharme, por compartir la vida conmigo.
Gracias Silvia por estar siempre que te necesito.
Gracias Yoyo por tu apoyo decidido, por tu cariño y paciencia, por el soporte que siempre me has brindado en todos los aspectos de mi vida.
Gracias Toño por acudir cuando se te necesita.
Gracias Beto por tu cariño, por tu presencia siempre constante en mi vida.
Gracias Raúl por tu cariño y abrazo sincero, después de algún tiempo por fin terminamos lo que iniciamos.

Mitzi y Fanny gracias por su cariño y apoyo.

A mi Director de Tesis el Dr. J. Aurelio Medina Ríos que con su guía, consejos acertados y apoyo sin condición a este proyecto de investigación se ha llegado a la culminación de esta etapa, por el empuje y aliento que me ha brindado para continuar en este camino tan desconocido que a veces llega a ser la Investigación.

Agradezco a mi Co-Director de Tesis el Dr. Juan Manuel García García por su disposición permanente para aclarar cualquier duda, por la aportación de las ideas que fueron la base de esta Tesis.

Al Dr. Antonio Ramos Paz por su apoyo incondicional para la consecución de esta meta profesional, simplemente no hubiera sido posible sin su apoyo. Por la paciencia que me ha tenido, por compartir su área de trabajo conmigo, pero por encima de todo, gracias por su amistad.

Al Dr. Juan José Flores Romero por sus consejos, observaciones y sugerencias aportadas, ha sido una experiencia gratificante trabajar con usted.

A mi compañero y amigo Ing. Guillermo Tapia Tinoco por su ayuda en los momentos difíciles que pasamos como alumnos. Eres una persona que tiene mi cariño y respeto.

Al M. C. Ignacio Antolino por tener siempre un tiempo para resolver una duda, por su ayuda incondicional.

Agradezco al Consejo Nacional de la Ciencia y Tecnología el apoyo económico que como becaria recibí y que me permitió llegar a la culminación de este proyecto.

Resumen

En esta Tesis se presenta el diseño e implementación de una base de datos distribuida haciendo uso de herramientas computacionales de código libre y la construcción de un cluster empleando computadoras personales.

Esta Tesis también reporta una comparación práctica de rendimiento entre dos maneras de implementar una base de datos distribuida. La primera, usando MySQL v.5.1 con el motor de almacenamiento NDB y la segunda, una combinación de Coda 6.9.3, y bases de datos particionadas de MySQL estándar. Se usaron dos diferentes ambientes de cómputo distribuido; de 4 y 10 nodos de datos, respectivamente. La comparación entre las plataformas MySQL Cluster y Coda está dada en términos del esfuerzo computacional requerido para manejar un gran número de registros de una base de datos distribuida, incorporando índices y tolerancia a fallos. Los detalles son dados en la implementación práctica del ambiente computacional de red, diseñado para un cluster distribuido.

Palabras Clave: Base de datos distribuida, motor de almacenamiento, cluster, MySQL, Coda.

Abstract

In this Thesis a design and implementation of a distributed database using open source computer tools and the cluster building using personal computers is presented

This Thesis also reports a practical performance comparison between two ways of implementing a distributed database. First, using MySQL v.5.1 with NDB Cluster storage engine, and second, a combination of Coda 6.9.3 and standard MySQL partitioned databases, respectively. Two different distributed computer environment were used; with 4 and 10 data nodes, respectively. The comparison between MySQL Cluster and Coda platforms is given in terms of the computational effort required to handle a large number of registers in a distributed database, incorporation of indexes and fault tolerance. Details are given on the practical implementation of the computer network environment designed distributed for a distributed cluster.

Keywords: Distributed databases, storage engine, cluster, MySQL, Coda.

Lista de Símbolos y abreviaturas

Atomicidad, Consistencia, Aislamiento y Durabilidad - *Atomicity, Consistency, Isolation and Durability*	ACID
Base de Datos de Red - Network Database	NDB
Base de Datos Distribuida	BDD
Compromiso de Dos Fases - *Two Phase Commit*	2PC
Gigabyte	GB
Infinito	∞
Instituto Nacional Americano de Estándares - *American National Standards Institute*	ANSI
Lenguaje de consulta estructurado - Structured Query Language	SQL
Máquina de Control del Sistema - *System Control Machine*	SCM
Megabit	Mb
Megabyte	MB
Memoria de Acceso Aleatorio - Random Access Memory	RAM
Millones de Instrucciones por Segundo	MIPS
Multiprocesamiento simétrico - *Symmetric Multi-Processing*	SMP
Número de muestra	N
Procesamiento Paralelo Masivo - *Massive Parallel Processing*	MPP
Protocolo de Control de Transmisión - *Transmission Control Protocol*	TCP
Sistema de Archivos de Red - Network File System	NFS
Sistema Distribuido de Bases de Datos	SDBD
Sistema Manejador de Bases de Datos	SMBD
Sistema Manejador de Bases de Datos Distribuidas	SMBDD
Tiempo	t
Unidad central de procesamiento - *Central Processing Unit*)	CPU

Lista de Figuras

Lista de Tablas

Lista de Publicaciones

1. **Una comparación de rendimiento entre dos maneras de implementar una base de datos distribuida usando MySQL**. Medina V., García-García J. M., Medina A., Ramos-Paz A. Artículo que será publicado en el 6° Congreso Internacional de Innovación y Desarrollo Tecnológico.

Contenido

Capítulo 1
Introducción

1 Introducción

Una base de datos distribuida es una colección de múltiples bases de datos interrelacionadas lógicamente a través de una red de computadoras. Por lo que un sistema manejador de bases de datos distribuidas (SMBD distribuidas) es definido como el software que permite el manejo de bases de datos distribuidas (BDD) y hace esta distribución transparente para los usuarios [Özsu y Valduriez, 1999].

Un SDBD está construido por un conjunto de nodos que están conectados en una red de computadoras. El contenido de las bases de datos debe estar relacionado, tener una estructura de archivos y una interfaz común de acceso. De esta manera, cualquier usuario, en cualquier parte de la red puede tener acceso a los datos, como si toda la información estuviera almacenada en el sitio local. Así la base de datos distribuida (BDD) puede ser almacenada en diferentes sitios físicamente, pero esto no significa que los sistemas están geográficamente lejos uno del otro, sólo que la comunicación entre ellos es por medio de la red, la cual es el único recurso compartido; incluso dos o más nodos pueden co-existir en la misma computadora.

Un sistema distribuido de archivos es un modelo extendido de un sistema de archivos individual, donde todos los usuarios de un conjunto de computadoras comparten el mismo sistema de archivos [Doeppner, 1996]. El usuario puede manejar la información en estos sistemas de archivos sin saber que los directorios que accede están localizados en diferentes máquinas, que la información está replicada o que un servidor apagado.

Un sistema de archivos distribuido almacena archivos en uno o más computadoras llamadas *servidores*, donde estos archivos se presentan al usuario como si fueran archivos locales. Hay varias ventajas en usar servidores de archivos: Los archivos tienen una alta disponibilidad debido a que varias computadoras pueden acceder los servidores y compartir los archivos desde una sola ubicación, lo cual es más sencillo que distribuir copias de archivos a clientes particulares. Los respaldos y la seguridad de la información son más fáciles de ejecutar, ya que solo los servidores deben ser respaldados. Los servidores pueden proporcionar un gran espacio de almacenamiento, lo que podría ser muy costoso o impráctico de proporcionar a cada cliente. La utilidad de los sistemas de archivos distribuidos se reconoce claramente cuando se pone en práctica con un grupo de empleados compartiendo documentos [Braam, 1998].

Los sistemas de archivos distribuidos han adquirido importancia en años recientes, pero la dependencia a esta clase de sistemas hace que el problema de disponibilidad se vuelva más importante. Actualmente, si un solo servidor se colapsa, puede traer muchos problemas a los usuarios y el principal de ellos la pérdida de su información, ésta es razón más importante por la cual un sistema de archivos distribuido puede ser de gran ayuda si el sistema de archivos proporciona la característica de alta disponibilidad.

1.1 Antecedentes

a) Base de Datos Distribuidas

Desde hace más de una década los sistemas de base de datos paralelos han estado desplazando a las máquinas mainframe para las grandes bases de datos y las tareas de procesamiento de transacciones. La mayoría de la investigación de las máquinas de bases de datos se ha concentrado en hardware especializado, a menudo de moda, tal como memorias CCD (siglas en

inglés del *charge-coupled device*), dispositivo de cargas eléctricas interconectadas, memorias de burbuja, discos cabeza-por-pista y discos ópticos [Banerjee *et al*., 1979] [De Witt, 1979] [De Witt y Gray, 1992]. Ninguna de estas tecnologías cumplió sus expectativas de eficiencia; por lo que hubo un sentido de que los CPU's convencionales, la RAM electrónica y los discos magnéticos dominarían la escena por muchos años. En ese tiempo, se predijo que el rendimiento de disco se duplicaría, mientras que la velocidad del procesador se incrementaría en factores más grandes. Consecuentemente, algunos críticos predijeron que los sistemas multiprocesador serían pronto limitados en entrada/salida a menos que fuera encontrada una solución al cuello de botella de entrada/salida.

Mientras que estas predicciones fueron bastante acertadas acerca del futuro del hardware, los críticos se equivocaron acerca del futuro en general de los sistemas de base de datos distribuidas. Durante la última década Teradata, Tandem y otras compañías han desarrollado y comercializado exitosamente máquinas de bases de datos distribuidas [De Witt y Gray, 1992].

Los sistemas de bases de datos distribuidas han encontrado aceptación porque se ha puesto atención en las siguientes situaciones:

- Primero, la mayoría de las grandes organizaciones están geográficamente descentralizadas y tienen sistemas computacionales múltiples en ubicaciones múltiples. Es impráctico tener un solo Administrador de Base de Datos (DBA, *Database Administrator*) para controlar los datos de una compañía alrededor del mundo; en lugar de ésto, es deseable tener un DBA en cada sitio y construir una base de datos distribuida que permita a los usuarios acceder los recursos de la compañía.

- Segundo, en ambientes con un alto nivel de transacciones se debe tener un equipo computacional robusto. Mientras es seguramente aceptable comprar una computadora mainframe (por ejemplo, una máquina de la clase Sierra IBM) será casi 2 órdenes de magnitud más barato ensamblar una red de máquinas más pequeñas y ejecutar un sistema de bases de datos distribuidas.

- Tercero, se supone que se desea la descarga de ciclos de la bases de datos de una gran mainframe a una máquina back-end. Si es así, tiene sentido soportar la posibilidad de más de un CPU back-end y un SMBD.

- Cuarto, se espera que más y más usuarios tengan computadoras en sus escritorios de trabajo y se espera que la mayoría de estas máquinas tengan sus propios discos para asegurar un buen rendimiento. En este ambiente, se tendrían un gran número de bases de datos en las estaciones de trabajo que podrían ser de naturaleza personal (tal como calendarios, directorios telefónicos, listas de correo, etc.) e incluso estas bases de datos personales requieren un sistema manejador de bases de datos distribuidas [Stonebraker,1989]. Tal es el caso de la programación electrónica o el correo electrónico empresarial que puede ser manejado con aplicaciones como Lotus Notes® de IBM [LOTUS, 2008].

Actualmente ha habido un continuo incremento de la cantidad de datos manejados por los Sistemas Manejadores de Bases de Datos. Es común que un SMBD maneje bases de datos con tamaños de cientos de gigabytes o terabytes. Este incremento masivo en los tamaños de las bases de datos es compaginado con la creciente necesidad de SMBDs para mostrar funcionalidades más sofisticadas, tal como el soporte para aplicaciones orientadas a objetos, deductivas y basadas en multimedia. En muchos casos, estos nuevos requerimientos han dado como resultado que los SMBDs existentes no proporcionen el rendimiento necesario del sistema, especialmente debido a que muchos SMBDs mainframe tienen dificultad para reunir los requerimientos de rendimiento de entrada/salida y de CPU necesarios para dar servicio a un gran número de usuarios concurrentes y/o que manejan cantidades de datos masivas [De Witt y Gray, 1992].

Para lograr los requerimientos de los niveles de desempeño, se ha requerido cada vez más que los sistemas de bases de datos hagan uso del paralelismo. El método tradicional para paralelizar SMBD convencionales, los cuales usan los modelos de base de datos estándar de la industria, tal como el relacional, puede tomar una de dos formas [Abdelguerfi y Wong, 1998]: La primera es a través del uso de plataformas paralelas de hardware de propósito general, por ejemplo, plataformas comerciales tales como nCube [NCUBE] y SP1 [Gropp y Lusk, 1995] son soportadas por el servidor paralelo de Oracle® [ORACLE]. El segundo método usa arreglos de componentes comerciales para formar sistemas paralelos masivos. La NCR 3700 [Witkowski et al. 1993] y la Super Database Computer II (SDC-II) [Tamura, *et al.* 1996] son dos de estos sistemas. La NCR 3700 usa una *red de interconexión multiestado*, que es una clase de redes de alta velocidad, empleada para conectar un conjunto de switches llamados switches de entrada a otros llamados switches de salida mediante una secuencia de estados de los switches [Blelloch y Maggs, 1996]. Este sistema puede ahora ejecutar una versión de un SMBD Sybase paralela.

La SDC-II es un servidor de base de datos relacional altamente paralelo, que consiste de ocho módulos procesadores de datos interconectados por dos redes, donde cada módulo consiste de hasta siete procesadores conectados por dos buses y cuatro manejadores de disco. La SDC-II emplea varias técnicas para soportar consultas eficientemente.

El uso de clusters de estaciones de trabajo como sistemas paralelos es un método más reciente que ya está teniendo impacto en la industria de los SMBDs [Abdelguerfi y Wong, 1998]. Estas redes de estaciones de trabajo proporcionan cantidades enormes de poder computacional agregado. Proporcionan un ambiente computacional de alto rendimiento viable y tienen varias ventajas sobre las grandes máquinas paralelas dedicadas, incluyendo costo, eliminación de un punto central de falla y escalabilidad. Su uso como máquina paralela virtual no excluye el uso de las máquinas individuales en más formas tradicionales. Por ejemplo, las versiones paralelas de SMBDs relacionales, tales como Oracle®, están ahora disponibles en clusters de sistemas compatibles con PC, proporcionando un alto rendimiento, a relativamente bajo costo.

El número de computadoras de bases de datos paralelas de propósito general o dedicado está incrementándose cada año. Es real prever que todos los sistemas manejadores de bases de datos de alto rendimiento en el año 2010 soportarán el procesamiento paralelo. El alto potencial de las bases de datos paralelas impulsa a vendedores, profesionales e investigadores a entender el concepto de sistemas de bases de datos paralelas [Abdelguerfi y Wong, 1998].

b) Sistemas de Archivos Distribuidos

El almacenamiento permanente es una abstracción fundamental en computación. Consiste de un conjunto nombrado de objetos que existen por creación explícita, son inmunes a fallas temporales del sistema y persisten hasta que sean eliminados explícitamente. El nombrado de la estructura, las características de los objetos y el conjunto de operaciones asociadas con éstos, caracteriza un refinamiento específico en la abstracción básica; un sistema de archivos es tal refinamiento.

Desde el punto de vista del diseño de un sistema de archivos, los modelos de computación pueden ser clasificados en cuatro niveles. El conjunto de características de diseño a cualquier nivel incluye las de niveles inferiores. Consecuentemente, la implementación de un sistema de archivos para un nivel más alto habrá de ser más sofisticada que una para un nivel más bajo.

1. En un nivel más bajo, un usuario en un solo sitio ejecuta cálculos vía un solo proceso. Un sistema de archivos de este modelo debe dirigirse por cuatro eventos clave. Estos incluyen la estructura de nombrado de un sistema de archivos, una interfaz de programación, el mapeo de la abstracción sistema de archivos al medio de almacenamiento físico y la integridad del sistema de archivos a través de fallas de corriente, hardware, medios y software.

2. El siguiente nivel, involucra a un usuario interactuando con múltiples procesadores en un sitio. Control de concurrencia es ahora una consideración importante en la interfaz de programación y la implementación del sistema de archivos.

3. El modelo clásico de tiempo compartido, donde múltiples usuarios comparten datos y recursos, constituye el tercer nivel de la taxonomía. Los mecanismos para especificar y asegurar la seguridad son importantes. Unix es el prototipo ideal del sistema de archivos de tiempo compartido.

4. Los sistemas de archivos distribuidos constituyen el nivel más alto de la taxonomía. Múltiples usuarios que están dispersos físicamente en una red de computadoras autónomas comparten el uso de un sistema de archivos común. El reto es realizar esta implementación de una manera segura y robusta. Además, las características de ubicación y disponibilidad toman gran importancia.

El método más simple para la ubicación de archivos es integrar la información de ubicación en los nombres. Pero el ligado estático de nombre a la ubicación dificulta la movilidad de los archivos entre sitios. También requiere que los usuarios recuerden los nombres de las máquinas, lo cual es complicado en un ambiente distribuido grande. Un método mejor es usar la transparencia de ubicación, donde el nombre del archivo no está incluido en la información de la ubicación. Un mecanismo de ubicación de archivos explícito mapea dinámicamente nombres de archivo a sitios de almacenamiento.

La *replicación* es una técnica básica usada para lograr alta disponibilidad en un sistema de archivos distribuido, aunque tiene sus complicaciones, ya que los cambios realizados deben ser propagados a todas las réplicas y éstas deben ser hechas de una forma consistente y eficiente [Satyanarayanan M., 1989].

1.2 Estado del Arte

a) Bases de Datos Distribuidas

Durante las últimas dos décadas se han publicado muchos trabajos en relación a las bases de datos distribuidas y/o paralelas y también han surgido varios productos que han sido relevantes como los que a continuación se mencionan.

Teradata

Teradata® ha sido el pionero de muchas ideas en el procesamiento de bases de datos distribuidas. Desde 1978 ha construido sistemas SQL (por sus siglas en inglés, Structured Query Language) altamente paralelos de arquitectura *shared-nothing* basados en microprocesadores, discos y memorias comerciales.

Los sistemas Teradata pueden tener más de mil procesadores y miles de discos. Los procesadores Teradata están funcionalmente divididos en dos grupos: Procesadores de Interfaz (Interface Processors, IFPs) y Procesadores de Módulo de Acceso (Access Module Processors, AMPs). Los IFPs manejan la comunicación con el host, *parsing* y optimización de consultas y la coordinación de AMPs durante la ejecución de la consulta. Los AMPs son responsables de ejecutar consultas. Cada AMP usualmente, tiene varios discos y una memoria caché grande. IFPs y AMPs están interconectados a través de una red llamada red-Y, redundante dual, en forma de árbol.

Cada relación es particionada por medio de una función *hash* [Cormen *et al.,* 2001] a través de subred de AMPs. Teradata ha instalado muchos sistemas conteniendo más de cien procesadores y cientos de discos. Estos sistemas demuestran *speed-up* [Mattson *et al.,* 2005] cerca de lo lineal y

un *scale-up* [Mattson *et al.,* 2005] en consultas relacionales, excede por mucho la velocidad de los *mainframes* tradicionales en cuanto a procesar bases de datos grandes (terabytes) [De Witt y Gray, 1992].

Tandem NonStop SQL

NonStop SQL es una implementación de ANSI SQL sobre Tandem Computer Systems [TERADATA]. NonStop SQL proporciona datos distribuidos y ejecución distribuida. Puede ejecutarse sobre computadoras pequeñas y se han registrado más de 200 transacciones por segundo en sistemas grandes. NonStop SQL proporciona alta disponibilidad a través de la combinación de un dispositivo de soporte y mecanismos de transacción NonStop. [Tandem Database Group, 1987].

NonStop SQL usa implícitamente SQL. NonStop es un SQL distribuido, de alto rendimiento. Tiene las características de rendimiento, integridad, administrativas y utilidades requeridas para soportar cientos de transacciones por segundo trabajando con cientos de gigabytes de bases de datos.

NonStop tuvo otras metas de diseño:

- Ser integrado con la red Tandem y el sistema de procesamiento de transacciones.
- Proporcionar a NonStop acceso a los datos, en caso de falla, solamente las transacciones afectadas son abortadas y reiniciadas; los datos siempre están disponibles.
- Soportar el crecimiento modular de hardware, y como consecuencia, soportar decenas de procesadores ejecutando cientos de transacciones por segundo.
- Permitir que los datos y la ejecución sean distribuidos a través de la red local

El reto fue integrar el lenguaje SQL con la preexistente arquitectura del sistema Tandem.

Algunas metas fueron excluidas de la primera versión. Hubo un pequeño intento por explotar la arquitectura en paralelo del sistema Tandem para obtener paralelismo dentro de una transacción en el estilo de Teradata; en vez de esto, el paralelismo es explotado teniendo múltiples transacciones independientes ejecutándose a la vez. Esta implementación no se enfoca en resolver el problema de la base de datos heterogénea.

El diseño de NonStop SQl proporciona una base excelente para una implementación SQL altamente paralela.

Arquitectura NonStop

La arquitectura de hardware de Tandem consiste de procesadores convencionales, cada uno con hasta 16Mb de memoria principal. Los procesadores no comparten memoria; redes locales duales de 20 Mb/seg pueden ser conectados hasta 16 procesadores. Una extensión de fibra óptica de 4 posiciones permite interconectar hasta 224 procesadores. La comunicación y los dispositivos controladores de disco son duales y adjuntados a dos procesadores, por lo que hay rutas duales para el almacenamiento y medios de comunicación.

El sistema Tandem NonStop SQL está compuesto de clusters de procesadores interconectados a través de anillos de fibra óptica de 4 posiciones. Los sistemas Tandem ejecutan las aplicaciones sobre los mismos procesadores y sistema operativo que los servidores de bases de datos. Los sistemas están configurados a un disco por MIPS (Millones de Instrucciones por Segundo), así,

cada diez MIPS tienen cerca de diez discos. Generalmente, los discos son duplicados; cada disco es servido por un conjunto de procesadores manejando un caché compartido grande, un conjunto de bloqueos y bitácoras de eventos ocurridos para los datos sobre un par de discos. Se destina un esfuerzo considerable en optimización de barridos secuenciales a través del pre-envío de grandes unidades, por el filtrado y manipulación de tuplas con predicados SQL en estos servidores de disco. Esto minimiza el tráfico de las interconexiones compartidas de disco.

Gamma

Gamma es una máquina de base de datos relacional que opera sobre un hipercubo Intel iPSC/2 con 32 procesadores y 32 manejadores de disco. Gamma emplea tres ideas técnicas llave que hacen posible que la arquitectura sea escalada a 100s de procesadores. La primera, todas las relaciones son particionadas horizontalmente a través de múltiples discos, habilitando que las relaciones puedan ser barridas en paralelo. Segunda, nuevos algoritmos paralelos basados en hashing son usados para implementar operadores relacionales complejos como join y funciones agregadas. Tercera, las técnicas de programación de flujo de datos son usadas para coordinar consultas multioperador. Usando estas técnicas es posible controlar la ejecución de consultas muy complicadas con una coordinación mínima [De Witt *et al.*, 1990].

Gamma proporciona al administrador de la base de datos tres alternativas de *declustering* cuando se crea una relación: round-robin, hash y particionada por rango [Ghandeharizadeh S., 1990].

Bubba

El prototipo Bubba fue implementado usando un multiprocesador de 40 nodos FLEX/32 con 40 discos [Boral *et al.*, 1990]. Aunque fue un multiprocesador de memoria compartida, Bubba fue diseñado como un sistema shared-nothing y la memoria compartida es solamente usada para el paso de mensajes. Los nodos son divididos en tres grupos: Procesadores de Interfaz, para comunicación con los procesadores de hosts externos y coordinar la ejecución de consultas, Repositorios Inteligentes, para almacenamiento de datos y ejecución de consultas, y Repositorios *Checkpoint* / *Logging*. Aunque Bubba usa particionamiento como mecanismo de almacenamiento (particionamiento hash y por rango) y mecanismos de flujo de datos, Bubba es único en varios sentidos. Bubba usó FAD [Bancilhon *et al*., 1987] en vez de SQL como su lenguaje interfaz. FAD es un lenguaje de programación persistente extendido-relacional. FAD proporciona soporte para objetos complejos a través de varios constructores de tipo incluyendo sub-objetos compartidos, primitivas de manipulación de datos orientadas a conjuntos y constructores de lenguaje tradicionales. El compilador de FAD es responsable de detectar operaciones que puedan ser ejecutadas en paralelo de acuerdo a como los objetos de datos que se acceden han sido particionados. Otra característica de Bubba es el uso de un mecanismo de almacenamiento de un solo nivel en el cual la base de datos persistente en cada nodo es mapeada a un espacio de memoria virtual de cada proceso ejecutándose en el nodo. Esto es en contraste con el método usual de archivos y páginas [De Witt y Gray, 1992].

Oracle®

Oracle surge a finales de los 70's bajo el nombre de Relational Software, Larry Ellison vio la oportunidad de trabajar con el modelo de bases de datos relacionales y comercializar la tecnología. Después de 30 años, Oracle es un proveedor líder mundial de software para administración de información y la segunda compañía de software más grande del mundo [ORACLE].

En 2003, Oracle Corporation lanza al mercado Oracle 10g, donde "g" viene de "Grid", donde se incorpora el manejo y administración de datos en *grids*, que son un conjunto de bases de datos

cuya administración de espacio, recursos y servicios se puede realizar como si se tratara de una sola base de datos.

La arquitectura de ejecución paralela de Oracle Database Release 2 tomó las ventajas del hardware – SMP, cluster o MPP, para garantizar un desempeño óptimo y continuo. La base de datos de Oracle Database Release 2 controla y balancea todas las operaciones paralelas, en base a los recursos disponibles, prioridades requeridas y carga actual de sistema.

Con la ejecución paralela dinámica de Oracle®, todos los datos son compartidos y la decisión de paralelizar y dividir el trabajo en unidades más pequeñas no es restringida a una distribución de datos estática predeterminada hecha durante el tiempo de configuración de la base de datos (creación).

Cada consulta tiene sus propias características de acceso, sus propios métodos de combinar tablas y procesamiento de diferentes porciones de datos. Consecuentemente, cada estatuto SQL pasa por un proceso de optimización y un proceso de paralelización después de que es analizado sintácticamente. Cuando los datos cambian, si existe un esquema de ejecución mejor o un plan de paralelización está disponible o bien, se agrega un nuevo nodo al sistema, Oracle puede automáticamente adaptarse a la nueva situación. Esto proporciona un alto grado de flexibilidad para paralelizar cualquier clase de operación:

- El grado de paralelismo es optimizado para cada consulta. A diferencia de un ambiente *shared-nothing*, donde ni la memoria ni el almacenamiento periférico es compartido con otros procesadores [Stonebraker, 1986], Oracle no requiere invocar a todos los nodos a acceder todos los datos durante una consulta [Baer, 2005].

- Las operaciones pueden ejecutarse en paralelo, usando uno, alguno, o todos los nodos de un *Real Application Clusters* [Baer, 2005], dependiendo de la carga actual de trabajo, las características y la importancia de la consulta.

Tan pronto como el estatuto es optimizado y paralelizado, todas las subtareas subsecuentes son conocidas. El proceso original se convertirá en la consulta coordinadora; los servidores de ejecución paralela (parallel execution servers, PX servers) son ubicados en un pool común de servidores de ejecución paralela en uno o más nodos y comienzan a trabajar en paralelo sobre la operación.

Como en una arquitectura shared-nothing, cada servidor de ejecución paralela que se encuentra dentro en una arquitectura shared-everything trabaja independientemente sobre un subconjunto de datos. Datos y funciones son transportados entre procesadores paralelos similares. Cuando se determina el plan paralelo de un requerimiento, cada servidor de ejecución paralela conoce su conjunto de datos y tareas y la comunicación inter-proceso es [Baer, 2005].

Sin embargo, a diferencia de la arquitectura shared-nothing, cada estatuto SQL ejecutado en paralelo es optimizado sin la necesidad de tomar en cuenta ninguna restricción de diseño físico de la base de datos. Ésto posibilita configuración óptima de datos en cada ejecución paralela, proporcionando igual o mejor escalabilidad y desempeño que las arquitecturas shared-nothing [Baer, 2005].

DB2

DB2 [Baru, *et al.*, 1995] es una base de datos paralela y distribuida que inició operando sobre AIX (*Advanced Interactive eXecutive*). En la actualidad se encuentran disponibles para otras arquitecturas y sistemas operativos tales como Unix, Linux o Windows. Funciona sobre una arquitectura shared-nothing y sobre un modelo de ejecución de transporte, proporcionando dos características importantes: Escalabilidad y capacidad. DB2 puede manejar bases de datos con

cientos de gigabytes de datos. De la misma manera, el modelo del sistema habilita bases de datos a ser fácilmente escalados con la adición de un procesador o de un disco. La arquitectura e implementación de DB2 proporciona la oportunidad de un desempeño muy alto en el procesamiento de consultas [Baru, *et al.*, 1995].

- La tecnología de optimización de consultas considera una variedad de estrategias de ejecución para diferentes operaciones y consultas, y utiliza su costo para elegir la mejor estrategia posible de ejecución.

- El ambiente a tiempo de ejecución es optimizado para reducir el costo de procesamiento, el costo de sincronización y el costo de transferencia de datos.

- Las propiedades de una transacción ACID son cumplidas de una manera eficiente en el sistema para que proporcione transacciones completas.

- Utilerías de carga, importación, reorganización de datos y creación de índices han sido estructuradas eficientemente para ejecutarse en paralelo.

- Una utilería de reorganización en paralelo (redistribución) se utiliza para corregir datos y desequilibrio en la carga de procesamiento, a través de los diferentes nodos del sistema.

BD2 maneja tres métodos diferentes para construir bases de datos paralelas de alto rendimiento, los cuales son:

- Shared-memory (shared everything, fuertemente acoplado)
- Shared-disk (compartiendo datos, débilmente acoplado)
- Shared-nothing (datos particionados)

MySQL Cluster

En la actualidad, MySQL es uno de los sistemas manejadores de bases de datos de código abierto SQL más populares. MySQL es desarrollado, distribuido y soportado por MySQL AB [MySQL 5.1, 2007]. MySQL Cluster proporciona un ambiente de alta disponibilidad y alta redundancia, diseñada para trabajar un ambiente computacional distribuido en memoria. Usa el motor de almacenamiento NDB cluster para tener varios servidores MySQL colaborando en un cluster. MySQL Cluster tiene una arquitectura share-nothing, de tal forma que cada miembro del cluster tiene su propia memoria y disco. Fue el manejador de bases de datos distribuidas seleccionado para realizar las pruebas de esta Tesis, por lo que en el Capítulo 3 se describe en detalle.

b) Sistemas de Archivos de Distribuidos

La transferencia de archivos iniciada por usuario (user-initiated file transfer) fue la primera forma de acceso de archivo remoto [Satyanarayanan M., 1989]. Aunque limitada, sirvió como un mecanismo para compartir datos en el principio de la computación distribuida.

Un mayor paso en la evolución de los sistemas de archivos distribuidos fue el reconocer que el acceso a los archivos remotos podía hacerse de manera semejante al acceso a los archivos locales. Esta propiedad, llamada *transparencia de la red*, implica que cualquier operación que puede ser ejecutada con un archivo local puede ser ejecutada con un archivo remoto.

Durante la década de 1975 a 1985 hubo un gran desarrollo de sistemas de archivos experimentales. Sistemas tales como Felix, XDFS, Alpine, Swallow y Amoeba exploraron las características de las transacciones atómicas y control de concurrencia sobre archivos remotos. Los sistemas de archivos Cambridge y CMU-CFS trabajaron en los temas de cómo la estructura de

nombrado de un sistema de archivos distribuido podría ser separado de su función como un repositorio de almacenamiento permanente, lo cual implicaba control de acceso, caching, y migración de archivos transparente hacia el dispositivo de almacenamiento. Cedar fue el primer sistema de archivos que trabajó prácticamente con caching de archivos completos [Satyanarayanan M., 1989].

Locus fue un sistema de referencia en dos sentidos. Primero, propuso transparencia de ubicación como criterio de diseño. Segundo, proponía replicación, junto con mecanismos de detección de inconsistencias, para lograr alta disponibilidad [Levy y Silberschartz, 1990].

La disminución del costo de CPU y memoria motivaron la investigación sobre estaciones de trabajo sin discos locales u otros dispositivos de almacenamiento permanente. La operación sin disco se demostró en sistemas como V y RVD. En la actualidad, el uso de los sistemas de archivos distribuidos se encuentra muy difundido [Satyanarayanan M., 1989].

Network File System de Sun

Desde su aparición en 1985, el *Network File System* (NFS) de *Sun Microsystems* [Sandberg,1986] [Satyanarayanan, 1989] ha sido ampliamente utilizado tanto en la industria como a nivel académico. El soporte para un gran número de usuarios ha influido en su uso en el medio académico. Algunos fabricantes incluyen NFS como una característica por omisión en sus productos.

Portabilidad y heterogeneidad son dos consideraciones importantes en el diseño de NFS. Aunque el modelo del sistema de archivo original está basado en Unix, NFS ha sido portado a otros sistemas operativos tales como PC-DOS. El protocolo NFS define una interfaz RFC que permite al servidor exportar archivos locales para acceso remoto. Detalles de diseño como caching, replicación, nombrado y garantía de consistencia pueden variar de una implementación a otra.

SunOS define un nivel de indirección en el kernel que permite que las operaciones del sistema de archivos sean interceptadas y ruteadas transparentemente hacia sistemas de archivos locales y remotos. Esta interfaz, llamada interfaz vnode, ha sido incorporada a muchas versiones de Unix.

Las estaciones de trabajo SUN pueden ser configuradas para trabajar sin disco local, la capacidad de estas estaciones de trabajo sin una degradación significativa de desempeño es otra meta de NFS. El NFS trata a las estaciones de trabajo como pares, una estación podría ser un servidor exportando sus archivos y podría ser cliente, accediendo archivos de otras estaciones de trabajo.

Generalmente, los clientes NFS usan una extensión del mecanismo *mount* de Unix, los subdirectorios exportados por servidores NFS son ligados a nodos del sistema de archivos *root*; esto ocurre cuando Unix es inicializado y permanece en efecto hasta que es modificado explícitamente. Ya que cada estación de trabajo es libre de configurar su propio nombre de espacio, no existe la garantía de que todos los usuarios tengan una vista común de sus archivos compartidos; por lo que la transparencia de ubicación se logra por convención y no como una característica de la arquitectura NFS.

Los clientes NFS guardan en caché páginas individuales de archivos y directorios remotos. También guardan en caché los resultados de la ruta del nombre a las transacciones vnode. Originalmente, NFS no soportaba replicación, pero versiones más recientes soportan replicación vía un mecanismo llamado *Automounter* [Satyanarayanan, 1989].

Andrew File System

Andrew es un ambiente de estaciones de trabajo que ha estado en desarrollo en la Universidad Carnegie Mellon desde 1983 [Howard *et al.,* 1988]. El mecanismo principal para compartir datos es un sistema de archivos distribuido que involucra todas las estaciones de trabajo. Usando un

conjunto de servidores de confianza, llamado colectivamente *Vice*, el sistema de archivos Andrew presenta un espacio de nombres de archivos de ubicación transparente y homogénea para las estaciones de trabajo.

El espacio de nombre de archivos en una estación de trabajo es particionado en un espacio de nombres compartido y un local. El espacio compartido es una ubicación transparente y es idéntica para todas las estaciones de trabajo. El espacio de nombres local es único a cada estación de trabajo y es relativamente pequeño. Solo contiene archivos temporales o archivos necesarios para la inicialización de la estación de trabajo. Los usuarios ven una imagen consistente de sus datos cuando se mueven de una estación a otra, debido a que sus archivos se encuentran en el espacio de nombres compartido.

El espacio de nombres compartido es particionado en subárboles separados y cada subárbol es asignado a un solo servidor, llamado su guardián. Cada servidor contiene una copia de una base de datos de ubicación que mapea archivos hacia sus guardianes.

Un manejador de caché, llamado Venus se ejecuta en cada estación de trabajo. Cuando un archivo es abierto, Venus verifica la caché para identificar la presencia de una copia válida. Si la copia existe, el requerimiento es tratado como una apertura de archivo local. De manera contraria, una copia actualizada es traída desde el guardián. Las operaciones de escritura y lectura son direccionadas hacia la copia en caché. Ningún tráfico de red es generado por estos requerimientos. Si un archivo en caché es modificado, este es copiado al guardián cuando el archivo es cerrado.

Un volumen en Andrew es un conjunto de archivos que forman un subárbol del espacio de nombres Vice y que tienen el mismo guardián. La creación de replicas de estos volúmenes pueden ser usadas para mejorar la disponibilidad y el rendimiento [Satyanarayanan M., 1989].

Coda

El Sistema de Archivos Distribuidos Coda es un sistema de archivos experimental desarrollado en el grupo de M. Satyanarayanan en Carnegie Mellon University (CMU). Coda posee las características [Braam P. J., 1998]:

- Tolerancia a fallas
- Desempeño y escalabilidad
- Seguridad
- Computación móvil
- Semánticas para compartir bien definidas
- Código fuente disponible libremente

Coda fue elegido para implementar una de las plataformas de prueba de la presente Tesis, por lo cual se hace una descripción a detalle de este sistema de archivos distribuido en el Capítulo 3.

Durante las últimas dos décadas, tanto en área de bases de datos como en las de sistemas de archivos se han liberado varias aplicaciones que se ejecutan en un ambiente distribuido; sin embargo, la aparición de nuevas tecnologías de hardware, tales como la multi-core, genera áreas donde los algoritmos actuales pueden ser mejorados, como es el caso del procesamiento distribuido de consultas o los protocolos de replicación distribuida.

1.3 Objetivos

Los objetivos principales de esta tesis de maestría son los siguientes:

1.3.1 Generales

- La implementación de una base de datos distribuida mediante el uso de la tecnología proporcionada por MySQL Cluster para un número arbitrario de nodos.

- La implementación de un sistema de archivos distribuido para un número arbitrario de volúmenes haciendo uso de Coda 6.9.3.

- Realización de pruebas de desempeño entre dos maneras de implementar una base de datos distribuida.

- Desarrollar experiencia local en la construcción y operación de un cluster de cómputo distribuido.

1.3.2 Particulares

- La construcción de un cluster de procesamiento distribuido a través de computadoras personales

- Comparación de eficiencia en tiempo de respuesta entre las dos plataformas de implementación de una base de datos distribuida.

1.4 Justificación

En base a lo expuesto en la revisión del estado del arte, durante las últimas dos décadas la investigación en el área de las bases de datos distribuidas y en sistemas de archivos distribuidos ha tenido gran auge. Tanto empresas que se dedican al desarrollo de software comercial, como el software de código abierto y experimental han incursionado en ambas áreas. Otros factores han favorecido el incremento de la tecnología distribuida, tales como la reducción del costo del hardware; el incremento en la complejidad de los datos manejados, tal como en las aplicaciones orientadas a objetos, deductivas o multimedia. Lo anterior lleva a que en la mayoría de las ocasiones se procese una gran cantidad de los datos, creando un ambiente propicio para que las aplicaciones sean desarrolladas en una plataforma de base de datos distribuidas y/o paralelas.

Para el desarrollo de la presente Tesis se propone una comparación práctica entre dos formas de realizar la implementación de una base de datos distribuida usando todas las facilidades técnicas que proporciona MySQL v. 5.1. y Coda 6.9.3. Se dividieron los Casos de Estudio en dos partes, considerando las dos plataformas propuestas; durante la primera parte, la base de datos distribuida fue conformada de 4 nodos de datos y durante la segunda, la base de datos se distribuyó en 10 nodos de datos. En la primera plataforma implementada se usó la tecnología MySQL Cluster para operar la base de datos distribuida y en la segunda, se utilizó de manera conjunta el sistema de archivos distribuidos Coda 6.9.3 y la base de datos particionada mediante MySQL estándar. Dichas implementaciones funcionan sobre un sistema operativo Ubuntu 7.0.4.

Se reporta la comparación entre las plataformas MySQL Cluster y Coda en términos del esfuerzo computacional requerido para manejar un gran número de registros de una base de datos distribuida, incorporando índices y el manejo de tolerancia a fallas.

Por último, se busca que la implementación realizada sea útil con fines académicos y de investigación.

1.5 Metodología

La metodología asociada con el desarrollo de esta Tesis se basa en la ejecución de las siguientes actividades:

1. **Revisión de arquitecturas distribuidas y esquemas de procesamiento distribuido y/o paralelo**. En esta etapa se realiza una revisión a detalle de las arquitecturas computacionales para procesamiento de bases distribuidas existentes y sus requerimientos de implementación
2. **Diseño, implementación y operación de una red distribuida de cómputo: hardware y software.** En esta etapa se procede a crear el diseño físico del cluster sobre el cual se implementará la base de datos distribuida. En esta fase se realiza la instalación del sistema operativo en las computadoras que conforman el cluster, se efectúa la instalación y configuración de MySQL v.5.1. habilitando la capacidad de manejar clusters, y además instalación y configuración del sistema de archivos distribuido Coda 6.9.3, de tal manera que todos los ambientes puedan co-existir y operar.
3. **Puesta en operación de la base de datos distribuida sobre dos plataformas diferentes.** Una vez que se ha diseñado y construido el ambiente adecuado para la operación de las plataformas para bases de datos distribuidas, se procederá a verificar su adecuada operación sobre cada una de ellas, así como los protocolos de comunicación, transferencia de información y la tolerancia a fallas.
4. **Casos de estudio y preparación de artículos especializados:** En esta etapa se procede a verificar la eficiencia del sistema de la base de datos implementada mediante casos de estudio computacionalmente demandantes. Se considera el análisis del tiempo de respuesta en el manejo y procesamiento de grandes bases de datos en ambas plataformas implementadas.

1.6 Descripción de Capítulos

En el Capítulo 1 se hace una reseña de los antecedentes históricos en las áreas de redes de cómputo distribuido, bases de datos distribuidas y sistemas de archivos distribuidos, así como la mención de los principales proyectos que se han desarrollado en dichas áreas y sus principales características.

En el Capítulo 2 se describe de manera detallada como se encuentra la configuración de las dos plataformas sobre las cuales se implementó la base de datos distribuida y se describe cada una de las pruebas de los Casos de Estudio.

En el Capítulo 3 se hace una descripción detallada de las características de las bases de datos distribuidas y de los sistemas de archivos distribuidos.

En el Capítulo 4 se realiza una descripción de las herramientas computacionales específicas utilizadas para la implementación de la base de datos distribuida.

En el Capítulo 5 se hace una descripción detallada de los Casos de Estudio realizados y de los resultados obtenidos en cada uno de ellos.

En el Capítulo 6 se ofrecen las conclusiones y aportaciones obtenidas del desarrollo de esta Tesis de maestría así como los trabajos de investigación que se pueden desarrollar a partir de esta investigación.

Capítulo 2
Dos Aplicaciones de los Sistemas Distribuidos

En este Capítulo de describe de manera general dos aplicaciones que se encuentran ampliamente difundidas en el ámbito del cómputo distribuido: Las bases de datos y los sistema de archivos distribuidos

2.1 Introducción

La tecnología de los sistemas de bases de datos distribuidas es la unión de dos métodos que parecen ser diametralmente opuestos al procesamiento de datos: Tecnologías de sistemas de bases de datos y redes de computadoras [Özsu y Valduriez, 1999].

Una de las mayores motivaciones detrás del uso de los sistemas de bases de datos es el deseo de integrar los datos operacionales de una empresa y de proporcionarlos centralizados, así como un acceso controlado a los datos. La tecnología de redes de computadoras, por otro lado, promueve una forma de trabajar que va en contra de los esfuerzos centralizados. A simple vista parece difícil comprender como integrar dos tecnologías basadas en metodología opuestas para producir una tecnología más poderosa y con mayores expectativas que cualquiera de las dos por si mismas. Y la clave esta en observar que el objetivo principal de la tecnología de bases de datos es la integración y no la centralización; ninguno de estos términos implica el otro. Es posible lograr la integración sin centralización, lo cual es el objetivo de la tecnología de bases de datos distribuidas [Özsu y Valduriez, 1999].

2.2 Bases de datos distribuidas

A continuación se dará una descripción de las arquitecturas que son utilizadas en la configuración y operación de las bases de datos distribuidas; así como de las características más relevantes que un sistema manejador bases de datos distribuidas considera en su diseño y algunos protocolos que intervienen para asegurar la integridad y la alta disponibilidad de la información.

2.2.1 Procesamiento distribuido

Se define un sistema computacional distribuido como un número de elementos de procesamiento autónomos (no necesariamente homogéneos) que están interconectados a través una red de computadoras y que cooperan para ejecutar sus tareas asignadas. El “elemento de procesamiento” referenciado está definido como un dispositivo computacional que puede ejecutar un programa por si mismo [Özsu y Valduriez, 1999].

Existen varios elementos computacionales que pueden ser distribuidos como:

- *El procesamiento lógico:* En la definición de un sistema computacional distribuido dada arriba implícitamente considera que el procesamiento lógico o el procesamiento de elementos son distribuidos.
- *La función:* Varias funciones de un sistema computacional podrían ser delegadas a varias piezas de hardware o software.
- *Los datos:* Los datos usados por un número de aplicaciones podrían ser distribuidas a un número de sitios de procesamiento.
- *El control:* El control de la ejecución de varias tareas podría ser distribuido en vez de ser ejecutado por un solo sistema computacional.

Desde el punto de vista de los sistemas de bases de datos distribuidas estos modos de distribución son todos necesarios e importantes.

Los sistemas de cómputo distribuidos pueden ser clasificados con respecto a varios criterios. Algunos de estos, son los siguientes: grado de acoplamiento, estructura de interconexión, independencia de componentes y sincronización de componentes.

Grado de acoplamiento, se refiere a la medida que determina que tan estrechamente están conectados los elementos de procesamiento. Esto puede ser medido como la proporción de la cantidad de datos intercambiados a la cantidad del procesamiento local desempeñado en ejecutar una tarea. Si la comunicación se realiza a través de una red de computadoras, existe un acoplamiento débil entre los elementos de procesamiento. Sin embargo, si los componentes, son compartidos, existe un acoplamiento fuerte. Los componentes compartidos pueden ser tanto memoria primaria como dispositivos de almacenamiento secundario. En la *estructura de interconexión* se tratan los casos que tienen una interconexión punto-a-punto entre elementos de procesamiento, a diferencia de aquellos que usan un canal de interconexión común. Los elementos de procesamiento podrían depender uno del otro, muy fuertemente en la ejecución de una tarea, o esta interdependencia podría ser mínima como el paso de mensajes al inicio de la ejecución y reportar resultados al final. La sincronización entre los elementos de procesamiento podría ser mantenida por significado síncrono o asíncrono. Nótese que algunos de estos criterios no son enteramente independientes. Por ejemplo, si la sincronización entre elementos de procesamiento es síncrona, se esperaría que los elementos de procesamiento fueran fuertemente independientes y que posiblemente trabajaran en forma fuertemente acoplada.

2.2.2 Sistemas de bases de datos distribuidas

Una base de datos distribuida es un conjunto de bases de datos interrelacionadas lógicamente a través de una red de computadoras. Por lo que un sistema manejador de bases de datos distribuidas (SMBDD), es definido como el software que permite el manejo de sistema de bases de datos distribuidas y hace esta distribución transparente para los usuarios [Özsu y Valduriez, 1999].

Un sistema de base de datos distribuidos está construido por un conjunto de nodos que están conectados en una red de computadoras, el contenido debe estar relacionado, tener una estructura de archivos y una interfaz común de acceso. De esta manera, cualquier usuario, en cualquier parte de la red puede tener acceso a los datos (lectura, escritura, etc.), como si toda la información estuviera almacenada en el sitio local. Así la base de datos distribuida (BDD) puede ser almacenada en diferentes sitios físicamente, pero esto no significa que los sistemas están geográficamente apartados, podrían compartir la misma habitación, lo cual implica que la comunicación entre ellos es por medio de la red, la cual es el único recurso compartido; incluso dos o más nodos pueden co-existir en la misma computadora [Özsu y Valduriez, 1999].

La definición antes mencionada reglamenta sistemas multiprocesador como sistemas de bases de datos distribuidas (SBDDs). Un sistema multiprocesador, ilustrado por la Figura 2.1 es un sistema donde dos o más procesadores comparten alguna forma de memoria, ya sea memoria primaria, en cuyo caso el multiprocesador es conocido como *memoria compartida* (fuertemente acoplada), o memoria secundaria, cuando se le denomina de *disco compartido* o débilmente acoplada, ilustrada en la Figura 2.2.

Otra distinción que es comúnmente hecha en este contexto es entre las arquitecturas *shared-everything* y *shared-nothing* [Özsu y Valduriez, 1996]. El modelo arquitectónico primero permite a cada procesador acceder todo (memorias primaria, secundaria y periféricos) en el sistema y cubre los dos modelos que se describen arriba. La memoria compartida habilita al procesador a comunicarse sin intercambiar mensajes.

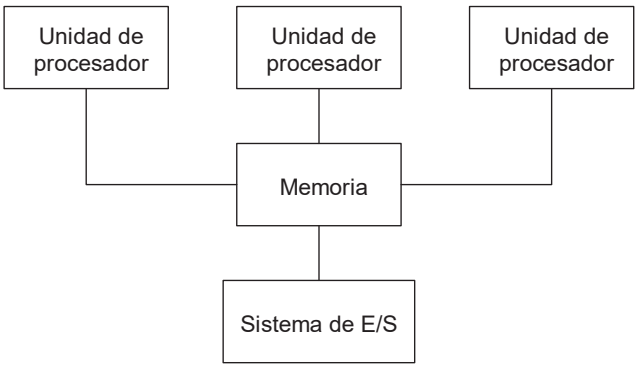


Fig. 2.1 Multiprocesador con memoria compartida

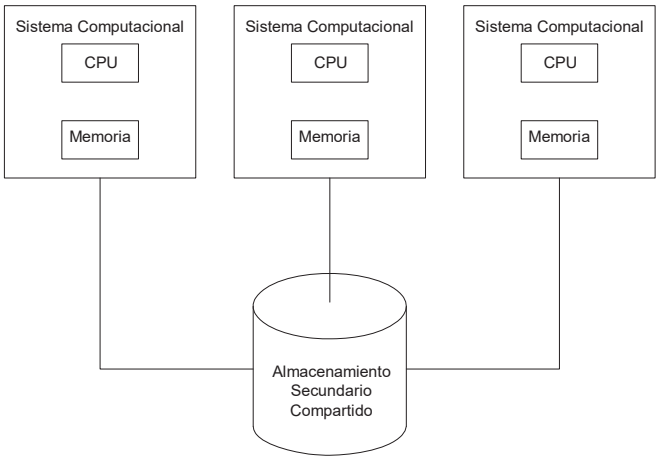


Fig. 2.2 Multiprocesador con disco compartido

La arquitectura *shared-nothing* es una donde cada procesador tiene sus propias memorias primaria y secundaria, así como periféricos y se comunica con otros procesadores a través de una interconexión de muy alta velocidad (por ejemplo, bus o switch), como se muestra en la Figura 2.3. En este sentido, los multiprocesadores *shared-nothing* son muy similares al ambiente distribuido. Sin embargo, hay diferencias entre las interacciones en arquitecturas multiprocesador y la débil interacción que es común en los ambientes de computación distribuida. La diferencia fundamental es el modo de operación; un diseño de sistema multiprocesador es bastante simétrico, consistiendo en un número de procesadores y componentes de memoria idénticos y controlados por una o más copias del mismo sistema operativo, el cual es responsable del control estricto de la asignación de tareas a cada procesador. Esto no es verdadero para los sistemas de cómputo distribuido, donde la heterogeneidad del sistema operativo, así como de hardware es común [Pinkerton *et al.,* 1990].

Además, un DDBS no es un sistema donde a pesar de la existencia de una red, la base de datos reside en un solo nodo. En este caso, los problemas del manejador de la base de datos no son diferentes a los problemas encontrados en un ambiente de base de datos centralizado. La base de datos es manejada centralmente por un sistema computacional y todos los requerimientos son ruteados a ese sitio. La única consideración adicional tiene que ver con retrasos de transmisión. Es obvio que la existencia de una red de computadoras o un conjunto de “archivos” no es suficiente para formar un sistema de bases de datos distribuidas. En lo que estamos interesados es en un ambiente donde los datos están distribuidos entre un número de sitios.

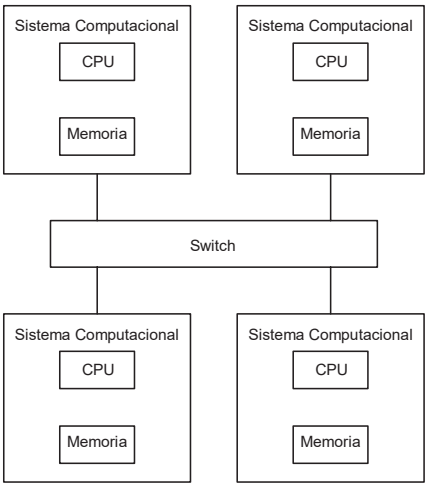


Fig. 2.3 Sistema multiprocesador *shared-nothing*

2.2.3 Elementos de un sistema de bases de datos distribuidas

Varias ventajas de los SBDDs han sido citadas en la literatura [Özsu y Valduriez, 1999], yendo desde razones sociológicas hasta económicas para descentralización. Todas estas pueden ser condensadas a cuatro fundamentales, que se citan a continuación.

Manejo transparente de datos distribuidos y replicados

Transparencia se refiere a la separación de semánticas de alto nivel de un sistema de los eventos de implementación de bajo nivel.
En otras palabras, un sistema transparente oculta los detalles de implementación a los usuarios. La ventaja de un SMBD completamente transparente es el alto nivel de soporte que proporciona el desarrollo de aplicaciones complejas. Al proceso donde se particiona alguna de las relaciones y almacenando cada partición en un sitio diferente; es conocido como *fragmentación*.

Además, podrían duplicarse algunos de estos datos en otros sitios por razones de desempeño y confiabilidad. El resultado es una base de datos distribuida, la cual es fragmentada y replicada. El acceso completamente transparente significa que el usuario puede realizar cualquier consulta sin tener conocimiento alguno de fragmentación, localización o replicación de datos, dejando que el sistema trate con estos eventos, para lo cual, el sistema debe tratar algunos tipos de transparencias.

Independencia de datos

La independencia de datos es una forma fundamental de transparencia que se ve dentro de un SMBD. Es también el único tipo que es importante dentro del contexto de un SMBD centralizado. Se refiere a la inmunidad de las aplicaciones de usuarios a cambios en la definición y organización de los datos y viceversa.

La definición de datos puede ocurrir en dos niveles: En un nivel la estructura lógica de los datos y en el otro nivel, la estructura física. El interior es conocido comúnmente como la *definición de esquema*, mientras que la última es referenciada como la *descripción física de datos*. Se puede hablar de dos tipos de independencia de datos: Independencia de datos lógica e independencia de datos física. Independencia de datos lógica se refiere a la inmunidad de las aplicaciones de usuario a cambios en la estructura lógica de la base de datos. En general, si una aplicación de usuario

opera sobre un subconjunto de los atributos de una relación, no debería ser afectado después, cuando los nuevos atributos son agregados a la nueva relación.

La independencia de datos física trata de esconder los detalles de estructura de almacenamiento de las aplicaciones de usuario. Los datos podrían estar organizados en diferentes tipos de discos, partes de esto podrían estar organizados diferentemente (por ejemplo, acceso aleatorio contra secuencial-indexado) o podría incluso ser distribuido a través de diferentes jerarquías de almacenamiento (almacenamiento a disco o cinta). Las aplicaciones de usuario no deberían ser modificadas cuando ocurren cambios en las estructuras de almacenamiento.

Transparencia de la red

En sistemas de bases de datos centralizadas, el recurso que debe ser protegido del usuario son los datos, por ejemplo, el sistema de almacenamiento. En un ambiente de manejador de bases de datos distribuidas, hay otro recurso que necesita ser protegido: La red. Preferentemente, el usuario desconoce los detalles operacionales de la red. Además, de ser posible, es preferible esconder la existencia de la red. Entonces, no habría diferencia entre las aplicaciones de bases de datos que se ejecutaran de manera centralizada y las que se ejecutaran sobre una base de datos distribuida. Este tipo de transparencia es llamada *transparencia de la red* o *transparencia de distribución*.

Desde la perspectiva de un SMBD, la transparencia de distribución requiere que los usuarios no tengan que especificar donde están localizados los datos.

Se ha clasificado la transparencia de distribución en dos tipos:

Transparencia de localización: Se refiere al hecho de que el comando usado para realizar una tarea es independiente de la localización de los datos y del sistema sobre el cual una operación es ejecutada.

Independencia de nombrado: Significa que un nombre único es asignado a cada objeto dentro de la base de datos. En la ausencia de transparencia de nombrado, los usuarios requieren integrar el nombre de la localidad (o un identificador) como parte del nombre del objeto.

Transparencia de replicación

Por razones de rendimiento, confiabilidad y disponibilidad, usualmente, es recomendable distribuir datos de una manera replicada a través de las máquinas que conforman una red. Tal replicación ayuda al rendimiento, ya que los diversos y complicados requerimientos del usuario pueden ser más fácilmente atendidos. La replicación permite que los datos que son comúnmente accedidos por un usuario, pueden ser grabados en la máquina local del usuario, así como en la máquina de otro usuario con los mismos requerimientos de acceso. Además, si una de las máquinas falla, una copia de los datos está aún disponible en otra máquina de la red. Sin embargo, la decisión de replicar o no y de cuantas copias de cualquier objeto de la base de datos se deben tener, depende en un grado considerable de las aplicaciones de usuario.

Uno de los problemas que la replicación debe resolver en la replicación de datos es la consistencia de las copias, donde el criterio de consistencia más usado asegura que todas las copias de un elemento de datos lógico debe de ser idénticas cuando una transacción de actualización termine [Özsu y Valduriez, 1996].

Asumiendo que los datos están replicados, un evento relacionado con la transparencia es, si los usuarios deben estar al cuidado de la existencia de copias o si el sistema debe manejar las copias y el usuario debería actuar como si solo hubiera una sola copia de los datos (no se hace referencia a la ubicación de las copias, solo a su existencia). Desde la perspectiva del usuario, es preferible

no estar involucrado en el manejo de las copias y tener que especificar que cierta acción necesita ser ejecutada sobre múltiples copias, sin embargo, si el usuario maneja las copias, el manejador de transacciones es más simple para los SMBDDs.

Finalmente, no es el sistema el que decide si tener o no copias, y cuantas tener, sino la aplicación de usuario.

Dadas las consideraciones anteriores, es deseable que la transparencia de replicación sea proporcionada como una característica estándar del SMBD. Se debe recordar que la transparencia de replicación se refiere solo a la existencia de réplicas, no a su ubicación actual y que distribuir estas réplicas a través de la red de una manera transparente esta en el dominio de la transparencia de la red.

Transparencia de fragmentación

La forma final de transparencia dentro del contexto de un sistema de bases de datos distribuidas es el de transparencia de fragmentación. Es deseable dividir cada relación de base de datos en fragmentos pequeños y tratar cada fragmento como un objeto base de datos separado (por ejemplo, otra relación). Generalmente, la fragmentación se hace por razones de desempeño, disponibilidad y confiabilidad. Además, la fragmentación puede reducir los efectos negativos de la replicación [Özsu y Valduriez, 1999]. Cada réplica no es una relación completa, solo un subconjunto de ella; por lo que se requiere menos espacio y pocos datos necesitan ser manejados.

Hay dos tipos de fragmentación, la *fragmentación horizontal*, donde una relación es particionada en un conjunto de subrelaciones, cada una de las cuales tiene un subconjunto de tuplas (filas) de la relación original. La segunda alternativa es la *fragmentación vertical*, donde cada subrelación es definida sobre un subconjunto de los atributos (columnas) de la relación original.

Cuando los objetos de la base de datos son fragmentados, se debe de tratar con el problema de manejar las consultas de usuario que se especifican para las relaciones enteras, ejecutadas en subrelaciones. Se debe encontrar una estrategia de procesamiento de consulta basada en los fragmentos, en vez de en las relaciones, aunque las consultas son definidas sobre relaciones. Se requiere de una traslación de una consulta global a varias consultas fragmento. La transparencia de fragmentación trata con el procesamiento de consultas.

La transparencia

Para proporcionar un acceso fácil y eficiente para los usuarios de los servicios de un SMBD, sería deseable la transparencia completa, involucrando todos los tipos que se mencionaron anteriormente. Por lo tanto, en el nivel de transparencia es inevitablemente un compromiso entre facilidad de uso y el costo elevado de proporcionar altos niveles de transparencia.

Es posible identificar tres capas distintas en las cuales los servicios de transparencia pueden ser proporcionados.

Se puede dejar la responsabilidad de proporcionar acceso transparente a recursos de datos a la capa de acceso. Las características de transparencia pueden ser construidas dentro del lenguaje del usuario, el cual convierte los servicios requeridos en operaciones requeridas. El compilador o el intérprete toma la tarea y el servicio no transparente se proporciona al desarrollador del compilador o interprete.

La segunda capa en la cual se puede proporcionar la transparencia es al nivel del sistema operativo. El estado del arte de los sistemas operativos proporciona algún nivel de transparencia a los usuarios del sistema. Por ejemplo, el usuario de computadora, o incluso un programador de

aplicación, generalmente, no desarrolla manejadores de dispositivo para interactuar con un equipo periférico individual; esta operación es transparente al usuario.

El acceso transparente a los recursos al nivel de sistema operativo, puede ser extendido a un ambiente distribuido, donde el manejo del recurso de la red es tomado por el sistema operativo distribuido.

La tercera capa en la cual la transparencia puede ser soportada es dentro del SMBD. Es responsabilidad del SMBD hacer todas las adecuaciones necesarias del sistema operativo a la interfaz de usuario de nivel más alto. Este modo de operación es el método más común por ahora. [Özsu y Valduriez, 1999].

La transparencia de red puede ser manejada fácilmente por el sistema operativo distribuido como parte de sus responsabilidades para proveer transparencias de replicación y fragmentación (especialmente aquellos aspectos tratando con manejo de transacción y recuperación). El SMBD debería ser responsable de proporcionar un alto nivel de independencia de datos junto con las transparencias de replicación y fragmentación. Finalmente, la interfaz de usuario puede soportar un alto nivel de transparencia, no solo en términos de un método de acceso uniforme a los recursos de los datos desde dentro de un lenguaje, sino también en términos de constructores de estructura que permiten al usuario tratar con objetos en su ambiente, en vez de enfocarse en los detalles de la descripción de la base de datos.

Jerarquía de transparencias

No siempre es fácil delinear claramente los niveles de transparencia, en la Figura 2.4 se muestra una idea de las capas de transparencia. Para completar la figura podemos agregar una capa de "transparencia de lenguaje". Con esta capa genérica, los usuarios tienen acceso de alto nivel a los datos (por ejemplo, lenguajes de cuarta generación, interfaces de usuario gráficas, acceso de lenguaje natural).

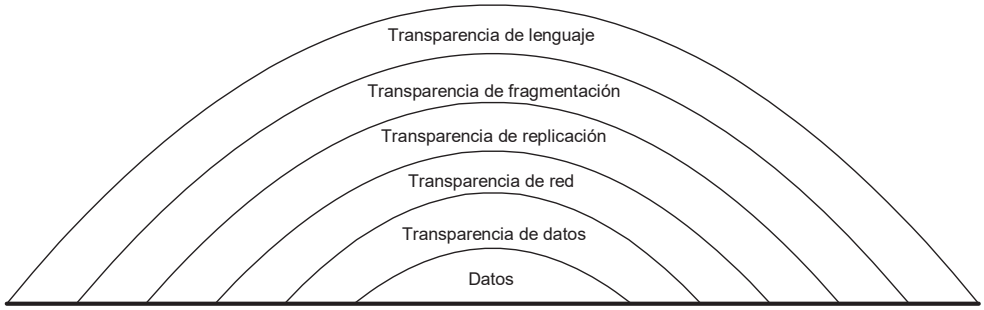


Fig. 2.4 Capas de transparencia

En la actualidad, la mayoría de los SMBD distribuidas comerciales, proporcionan algún nivel de soporte de transparencia. Usualmente, los sistemas proporcionan transparencia de distribución, soporte para fragmentación horizontal y alguna forma de transparencia de réplica; hasta recientemente, la mayoría de los SMBDs distribuidas comerciales no proporcionaban un nivel suficiente de transparencia. Algunos requieren que los usuarios integren la ubicación dentro del nombre de cada objeto de la base de datos. Además, requieren que el usuario especifique el nombre completo para el acceso al objeto.

En este punto es importante señalar que la transparencia completa no es un objetivo aceptado universalmente. Se argumenta que la transparencia completa hace el manejo de datos distribuidos muy difícil y asegura que "las aplicaciones codificadas con acceso transparente a bases de datos distribuidas geográficamente tienen: pobre manejabilidad, pobre modularidad y pobre desempeño de mensajes" [Gray, 1989]. Gray propone un mecanismo de llamada de un procedimiento remoto entre los usuarios requisitores y el servidor SMBDs, donde los usuarios direccionan sus consultas a

un DBMS específico. El manejo de datos distribuidos es más difícil si el acceso transparente es proporcionado por los usuarios y la arquitectura cliente/servidor como una comunicación basada en llamadas a procedimiento remoto entre los clientes y los servidores es el método arquitectónico correcto [Özsu y Valduriez, 1999].

Confiabilidad a través de transacciones distribuidas

Los SMBD distribuidas son destinados para mejorar la confiabilidad, debido a que han replicado componentes y por lo tanto, eliminado puntos de falla. La falla de un solo sitio, o la falla de una liga de comunicación que hace uno o más sitios inalcanzables, no es suficiente para colapsar el sistema completo. En el caso de una base de datos distribuida, esto significa que algunos de los datos podrían ser inaccesibles, pero con el cuidado apropiado, los usuarios podrían acceder otras partes de la base de datos distribuida. El "cuidado apropiado" entra en la forma de soporte para transacciones distribuidas y protocolos de aplicación. Una *transacción* es una unidad básica de computación consistente y confiable, conformada de una secuencia de operaciones de base de datos ejecutadas como una acción atómica, que transforma un estado consistente de la base de datos a otro estado consistente de la base de datos, aún cuando un número determinado de estas transacciones sean ejecutadas concurrentemente (algunas veces llamada transparencia de concurrencia) y aún cuando las fallas ocurran (también llamada falla de atomicidad). Por lo tanto, un SMBD que proporciona garantías de soporte de transacciones de usuario, no violarán la consistencia de la base de datos frente a las fallas del sistema, mientras la transacción sea correcta.

Rendimiento mejorado

El caso para rendimiento mejorado de SMBDs distribuidas es hecho típicamente basado en dos puntos:

1.- Un DBMS distribuido fragmenta la base de datos conceptual, habilitando datos para ser almacenados en una proximidad a sus puntos de uso (también llamados localización de datos). La mayoría de los SMBDs distribuidos son estructurados para obtener el máximo beneficio de la localización de datos; se pueden obtener beneficios completos de contención reducida y un costo comunicación reducido con una fragmentación y distribución de la base de datos apropiados.

2.- El paralelismo de sistemas distribuidos podrían ser explotados por paralelismo *inter-query* e *intra-query*. *Inter-query* que es el paralelismo que permite que múltiples consultas se ejecuten al mismo tiempo mientras que el paralelismo *intra-query* es logrado dividiendo un solo query en un número de subqueries, cada uno de los cuales es ejecutado en un sitio diferente, accediendo una parte diferente de la base de datos distribuida [Olson *et al.*, 1996].

El primer punto se relaciona con el costo de la computación distribuida si los datos tienen que residir en sitios remotos y se tienen que acceder por teleprocesamiento. Analiza la funcionalidad en el manejo de los datos y argumenta que es mejor distribuir los datos a donde sean localizados fácilmente, en vez de mover grandes cantidades de información.

También se argumenta que con el uso de la alta velocidad y alta capacidad de las redes, las funciones de distribución de datos y las funciones de manejo de datos ya no tienen sentido y podría ser más simple almacenar datos en un sitio central y accederlo (por descarga) a través de las redes de alta velocidad. Sin embargo, existen argumentos que apelan al anterior punto de vista, ya que no se distingue entre ancho de banda (la cantidad de datos que se pueden llevar de un punto a otro en un período dado*)* y latencia (cuanto tiempo toman los datos en ser transmitidos). La latencia es inherente en los sistemas distribuidos y hay limitantes físicas (como tipo de canal de transmisión, ancho de banda, etc.) de que tan rápido se pueden enviar datos a través de una red de computadoras.

El argumento del paralelismo también es importante. Si el usuario accede a la base de datos distribuida solamente por consultas (por ejemplo, acceso de solo lectura), entonces la provisión de paralelismo *inter-query* e *intra-query* implicaría que la mayor cantidad posible de la base de datos debería ser replicada. Sin embargo, debido a que la mayoría de los accesos a la base de datos no son de solo lectura, la mezcla de operaciones de lectura y actualización requiere de elaborados protocolos de control de concurrencia y commit.

Además de optimizar los sistemas para tratar con estos eventos, algunos sistemas comerciales toman un método muy interesante para tratar con los conflictos entre rendimiento de solo lectura y rendimiento de actualizaciones. Ellos multiplexan la base de datos manteniendo dos copias. Una copia es para consulta ad-hoc (llamada la *query database*) y la otra para actualización por programas de aplicación (llamada la *production database*). En intervalos regulares, la production database es copiada a la query database. Esto no elimina la necesidad de implementar protocolos de control de concurrencia y confiabilidad para la *production database* [Özsu y Valduriez, 1999].

Además de esto, hay una medida administrativa que algunos toman para tratar con el costo de manejo de transacciones. Algunas aplicaciones son configuradas para realizar consultas durante las horas de jornada laboral (por ejemplo, acceso de solo lectura), mientras que las operaciones de actualización son realizadas por medio de procesos *batch* en un horario fuera de la jornada laboral, entonces la base de datos se cierra a las consultas.

2.2.4 Conceptos fundamentales en la implementación de una base de datos distribuida

Speed-up y Scale-up

Un sistema paralelo ideal tiene dos propiedades principales [De Witt y Gray, 1992]:

1. *Speed-up* Lineal: El doble de hardware puede ejecutar la tarea en la mitad del tiempo transcurrido.
2. *Scale-up* Lineal: El doble de hardware puede ejecutar una tarea dos veces más grande en el mismo tiempo transcurrido.

Dado un trabajo fijo ejecutado en un sistema pequeño, y después ejecutado en un sistema grande, el speed-up dado por el sistema grande es medido como:

$$S = \frac{Ts}{Tb} \qquad (2.1)$$

donde,

S = Speed-up
Ts = Tiempo que toma el sistema en procesar un trabajo en una sistema pequeño
Tb = Tiempo que toma el sistema en procesar un trabajo en una sistema grande

El *speed-up* es lineal, si un sistema N-veces grande o más costoso lleva a un speed-up de N.

Speed-up mantiene el problema de tamaño constante y considera que el sistema crece en capacidades. *Scale-up* mide la disponibilidad de crecer tanto del sistema como del problema. *Scale-up* es definido como la habilidad de un sistema N-veces más grande de ejecutar un trabajo N-veces más grande en el mismo tiempo que el sistema original. La métrica *scale-up* es:

$$s = \frac{ts}{tb} \qquad (2.2)$$

donde,
s = scale-up
ts = Tiempo que toma un sistema de capacidades pequeñas en procesar un problema pequeño

tb = Tiempo que toma un sistema de capacidades grandes en procesar un problema grande

Si la ecuación *scale-up* es normalizada a 1, se dice que el *scale-up* es lineal.

Hay elementos que pueden impedir la existencia de *speed-up* lineal y *scale-up* lineal que son:

Startup: El tiempo requerido para comenzar una operación paralela. Si miles de procesadores debieran ser iniciados, esto puede dominar el tiempo de computación.
Interferencia: El retraso que cada nuevo proceso se impone sobre los otros cuando se accedan recursos compartidos.
Sesgo: Como el número pasos paralelos se incrementa, el promedio de cada paso se decrementa, pero la varianza puede exceder la media. El tiempo de servicio de un trabajo es el tiempo de servicio del paso más lento. Cuando la varianza domina la media, el paralelismo mejora el tiempo consumido solo ligeramente.

Computadoras con arquitectura shared-nothing.

La máquina ideal para una base de datos tiene un procesador infinitamente rápido, con una memoria infinita y un ancho de banda infinito – y sería infinitamente barata (gratis). Dada la existencia de solo una máquina, no sería necesario el paralelismo. La tecnología promete procesadores rápidos, discos rápidos de alta capacidad y memorias RAM de alta capacidad. También promete que estos dispositivos serán de bajo costo en comparación con los estándares actuales [De Witt y Gray, 1992].

Stonebraker, sugiere la siguiente clasificación para construir un sistema multiprocesador [Stonebraker, 1986]:

- *Memoria compartida:* Múltiples procesadores comparten acceso directo a una memoria central común.

- *Discos compartidos:* Múltiples procesadores cada uno con memoria privada comparten una colección común de discos.

- *Shared-nothing:* Ni la memoria ni el almacenamiento periférico es compartido entre los procesadores.

Las arquitecturas *shared-nothing* minimizan la interferencia, minimizando los recursos compartidos. También explotan los procesadores y memorias comerciales sin requerimientos altos de interconexión de red. Como la Figura 2.5 lo sugiere, las otras arquitecturas mueven grandes cantidades de datos a través de la interconexión de red. Los diseños *shared-nothing* mueven solamente preguntas y respuestas a través de la red. Los accesos a memoria y los accesos a disco son ejecutados localmente en un procesador y solamente los datos filtrados son transferidos al programa cliente. Esto permite un diseño más escalable minimizando el tráfico sobre la interconexión de red [De Witt y Gray, 1992].

Shared-nothing caracteriza los sistemas de bases de datos usados por Teradata, Gamma, Tandem, Bubba, Arbre y nCUBE. VAXcluster de Digital ha evolucionado a este diseño. Sistemas de trabajo DOS y UNIX de 3com, Borland, Digital, HP, Novell, Microsoft y Sun también han adoptado arquitecturas cliente-servidor *shared-nothing*.

La principal ventaja de los multiprocesadores *shared-nothing* es que pueden ser escalados a cientos y probablemente miles de procesadores que no interfieren unos con otros.

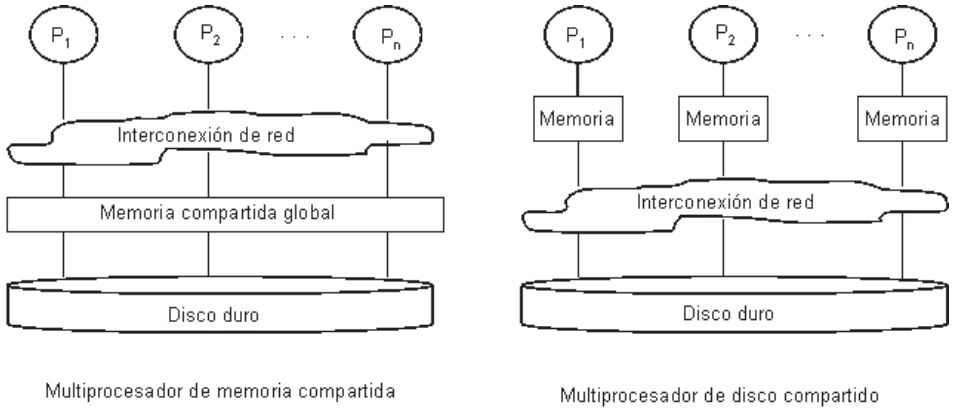


Fig. 2.5 Los diseños de memoria compartida y disco compartido

Estas arquitecturas *shared-nothing* logran speed-ups y scale-ups cerca de lo lineal en consultas relacionales complejas y en procesamiento de cargas en línea.

En la actualidad se han publicado algunos trabajos señalando las ventajas y desventajas de los sistemas multiprocesador [Stonebraker, 1986] [Thakkar y Sweiger, 1990], y donde se ha hecho notar que la arquitectura shared-nothing es escalable y altamente adecuada para que las aplicaciones de bases de datos tengan un rendimiento adecuado, sin presentar desventajas en comparación con otras arquitecturas alternativas.

Los arquitectos de las computadoras han tardado en adoptar la arquitectura *shared-nothing* debido a que los componentes comerciales de bajo costo, pero de alto rendimiento se han puesto a disposición en el mercado recientemente. Tradicionalmente, estos componentes fueron relativamente de bajo desempeño y de baja calidad.

Actualmente, el software es la barrera más significativa para usar el paralelismo. El software escrito para uniprocesadores no alcanza speed-up o scale-up cuando se instala en multiprocesadores. Debe ser reescrito para beneficiarse del procesamiento paralelo y de los discos múltiples. Las aplicaciones de base de datos son la única excepción a esto. Actualmente, la mayoría de los programas para bases de datos están escritos en el lenguaje relacional SQL que ha sido estandarizado por ANSI e ISO. Es posible tomar aplicaciones SQL estándar escritas para sistemas uniprocesador y ejecutarlas en paralelo o en máquinas de bases de datos *shared-nothing*. Los sistemas de bases de datos pueden distribuir datos automáticamente entre múltiples procesadores.

Particionamiento de datos

Particionar una relación involucra distribuir sus tuplas a través de varios discos. El particionamiento de datos tiene su origen en los sistemas centralizados donde se tenían que particionar archivos, ya sea porque el archivo era muy grande para el disco o porque la tasa de acceso a disco no podía ser soportada por un solo disco. Las bases de datos distribuidas usan particionamiento de datos cuando colocan fragmentos de la relación en diferentes sitios de la red. El particionamiento de datos permite a los sistemas de bases de datos paralelas explotar el ancho de banda de entrada/salida de múltiples discos, leyendo y escribiendo a ellos en paralelo. Este método proporciona un ancho de banda de entrada/salida superior a los sistemas de estilo RAID sin necesidad de hardware especializado.

La estrategia más simple de particionamiento distribuye las tuplas entre los fragmentos en forma *round-robin* [De Witt y Gray, 1992]. Esta es la versión particionada de un archivo de secuencia de entrada. El particionamiento *round-robin* es excelente para las aplicaciones que requieren acceder la relación por medio de un barrido secuencial en cada consulta. El problema del particionamiento *round-robin* es que frecuentemente, las aplicaciones requieren acceso de manera asociativa a las

tuplas, lo cual significa que las aplicaciones requieren encontrar las tuplas que tienen un valor de atributo en particular.

El particionamiento *hash* [De Witt y Gray, 1992] es ideal para aplicaciones que requieren solo de acceso secuencial y asociativo a los datos. Las tuplas son ubicadas aplicando una función hash a un atributo de cada tupla. La función especifica la ubicación de la tupla en un disco particular. El acceso asociativo a las tuplas con un valor específico de atributo puede ser direccionado a un solo disco, evitando el costo de empezar consultas en discos múltiples. Los mecanismos de particionamiento hash son proporcionados por Arbre, Bubba, Gamma y Teradata.

Hashing tiende a colocar de manera aleatoria los datos en vez de agruparlos. El particionamiento en rangos agrupa las tuplas con atributos similares en la misma partición. Esto es bueno para acceso secuencial y asociativo y también es bueno para clustering de datos. La Figura 2.6 muestra particionamiento de rangos basado en orden lexicográfico, pero cualquier algoritmo de agrupamiento es posible.

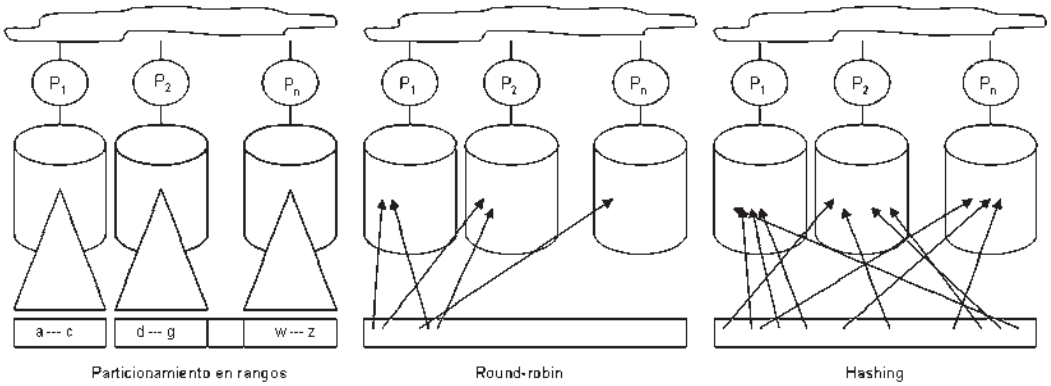


Fig. 2.6 Los tres esquemas básicos de particionamiento

El problema con el particionamiento de rango es que corre el riesgo de sesgo de datos, donde todos los datos son colocados en una partición y sesgo de ejecución, en la cual la ejecución ocurre en una partición. *Hashing* y *round-robin* son menos susceptibles a estos problemas de sesgo. El particionamiento de rango puede minimizar el sesgo escogiendo criterios de particionamiento no distribuidos uniformemente. Algunas aplicaciones usan este concepto considerando la frecuencia de acceso de cada tupla cuando se crean particiones en una relación; la meta es balancear la frecuencia con la cual cada partición es accedida en vez del número de tuplas en cada disco.

El particionamiento trae nuevas cuestiones físicas de diseño de las bases de datos. Cada relación debe tener una estrategia de particionamiento y un conjunto de fragmentos de disco. Incrementando el grado de particionamiento, usualmente se reduce el tiempo de respuesta para una consulta individual e incrementa el rendimiento total del sistema. Para barridos secuenciales, el tiempo de respuesta se decrementa debido a que más procesadores y discos son usados para ejecutar la consulta. Para barridos asociativos, el tiempo de respuesta mejora, porque pocas tuplas son almacenadas en cada nodo y por lo tanto el tamaño del índice que debe ser buscado decrementa.

2.2.5 Protocolos en bases de datos distribuidas

Control de Concurrencia

Siempre que exista el acceso de múltiples usuarios (lectura y escritura) a una base de datos compartida, estos accesos necesitan ser sincronizados para asegurar la consistencia de la base de datos. La sincronización se alcanza por medio de algoritmos de control de concurrencia, los cuales hacen cumplir criterios de corrección como la **serializabilidad**. Los accesos de usuario son

encapsulados como transacciones, cuyas operaciones al más bajo nivel son un conjunto de operaciones de lectura y escritura a la base de datos. Los algoritmos de control de concurrencia aseguran la propiedad de **aislamiento** de la ejecución de la transacción, la cual determina que los resultados de una transacciones hasta que la primera complete su ejecución. [Özsu y Valduriez, 1996].

Los algoritmos de control de concurrencia más populares son los basados en bloqueo. En tales esquemas, se coloca un bloqueo en alguna unidad de almacenamiento (generalmente una página), ya sea en modo compartido o exclusivo, siempre que una transacción intente accederla. Estos bloqueos se colocan para evitar conflictos de lectura-escritura, escritura-lectura y escritura-escritura. Las acciones de bloqueo siguen una regla, para asegurar la serializabilidad de las transacciones: "Ningún bloqueo por parte de una transacción debe ser colocado una vez que previamente se colocó y se liberó por la transacción". Este es conocido como bloqueo de dos fases, debido a que las transacciones van de una fase donde obtienen los bloqueos a una fase donde se liberan dichos bloqueos. En general, liberar los bloqueos antes del final de la transacción es problemático. La mayoría de los algoritmos de control de concurrencia basados en bloqueo son estrictos y mantienen sus bloqueos hasta el final de la transacción.

Protocolos de Confiabilidad

Los SMBD distribuidas son más confiables debido a que hay réplicas de cada componente del sistema, cada uno elimina puntos de falla. Ésto requiere la implementación de protocolos para tratar con fallas del sistema.

En un SMBD distribuidas hay cuatro tipos de fallas posibles:

- De transacción
- De sitio (sistema)
- De medio (disco)
- De la línea de comunicación

Las transacciones pueden fallar por varias razones. La falla puede ser debido a un error en la transacción causado por la entrada de datos, así también por la detección de un *punto muerto*, que es una condición donde dos o más procesos están esperando que otro proceso libere el recurso o más de dos están esperando recursos en una cadena circular. El método usual a tomar en casos de fallas de transacción es abortar la transacción, reiniciando la base de datos al estado anterior al inicio de la base de datos.

Las fallas de sitio (sistema) son debido a una falla de hardware (por ejemplo, procesador, memoria principal, fuente de energía) o una falla de software (*bugs* en el sistema o en el código de una aplicación). El efecto de las fallas de sistema es la pérdida del contenido de la memoria principal. Por lo tanto, cualquier actualización a las partes de la base de datos que se encuentre en los buffers de la memoria principal (también llamada base de datos volátil) es perdida como resultado de las fallas de sistema. Sin embargo, la base de datos que es guardada en almacenamiento secundario (también llamada base de datos estable) esta segura y correcta. Para lograr ésto, los SMBDs típicamente emplean protocolos de acceso, tal como *Write-Ahead Logging*, el cual graba los cambios a la base de datos en la bitácora del sistema y mueve estos registros y las páginas de la base de datos volátil al almacenamiento estable en tiempos apropiados.

Las fallas de medio se refieren a la falla de dispositivos de almacenamiento secundario que almacenan la base de datos estable. Usualmente, estas fallas son tratadas duplicando los dispositivos de almacenamiento y manteniendo copias de la base de datos. Las fallas de medios son frecuentemente tratadas como problemas locales de un sitio y por lo tanto son tratadas por los mecanismos de confiabilidad de los SMBDs distribuidas.

Los tres tipos de fallas descritos arriba son comunes en SMBDs centralizados y distribuidos. Las fallas de comunicación, por otro lado, son únicas de los sistemas distribuidos. Hay varios tipos de fallas de comunicación. Las más comunes son los errores en el mensaje, mensajes emitidos en orden incorrecto, pérdida de mensajes (no entregados) o fallas de línea. Generalmente, los primeros dos tipos de falla son considerados como responsabilidad de los protocolos de red y no son tratadas por el SMBD distribuidas. Las últimas dos, por otro lado, tienen un impacto en los protocolos del SMBD distribuidas, por lo que, son consideradas en el diseño de estos protocolos.

Las transacciones deben de cumplir algunas propiedades para conservar la integridad de los datos, tales como la propiedad de *atomicidad*, la cual asegura que todas o ninguna de las transacciones de actualización sean ejecutadas y la *durabilidad* [Frank L., 1999], la cual asegura que la transacción no pueda ser perdida después de que es comprometida, las transacciones durables son guardadas en un almacenamiento durable y aseguradas por un sistema de recuperación de bitácoras; tales propiedades requieren de la implementación del protocolo de compromiso atómico y de protocolo de recuperación distribuida. El protocolo de compromiso atómico más conocido es el de compromiso de dos fases.

El protocolo de dos fases (Two-Phase Commit, 2PC) es un protocolo simple y que asegura el compromiso atómico de las transacciones distribuidas. Extiende los efectos de las acciones atómicas locales a las transacciones distribuidas, insistiendo en que todos los sitios involucrados en la ejecución de una transacción acuerdan realizar el ***commit*** de una transacción antes de que sus efectos sean hechos permanentes (por ejemplo, todos los sitios terminan la transacción en la misma forma). Si todos los sitios acuerdan realizar el *commit* de una transacción, entonces todas las acciones de la transacción distribuida toman efecto; si uno de los sitios declina el *commit* de las operaciones en ese sitio; entonces a todos los otros sitios se les requiere abortar la transacción. Así, la regla fundamental de 2PC declara:

1. Incluso si un sitio rechaza realizar un commit de la transacción (lo cual significa que vota por abortar), la transacción distribuida tiene que ser abortada en cada sitio donde se ejecuta y;

2. Si todos los sitios votan por realizar el commit de una transacción; la transacción distribuida es realizada en cada sitio donde se ejecuta.

La ejecución del protocolo 2PC se realiza de la siguiente forma: Hay un proceso coordinador en el sitio donde se origina la transacción distribuida y los procesos participantes de los otros sitios donde la transacción se ejecuta. Inicialmente, el coordinador envía un mensaje de “preparación” a todos los participantes cada uno de los cuales determina independientemente si o no puede realizar la transacción *commit* en el sitio. Aquellos que pueden realizar un *commit* envían un mensaje “*vote-commit*”, mientras que aquellos que no pueden realizar un *commit* envían un mensaje de “v*ote-abort*”. Una vez que el participante registra su voto, no puede cambiarlo. El coordinador colecta estos mensajes y determina el destino de la transacción de acuerdo a la regla 2PC. Si la decisión es realizar un *commit*, el coordinador envía un mensaje de “*global-commit*” a todos los participantes; si la decisión es abortar, se envía un mensaje de “*global-abort*” a todos aquellos participantes que antes habían votado por realizar el *commit* de la transacción. El mensaje no necesita ser enviado a aquellos participantes que originalmente habían votado por abortar ya que pueden asumir, de acuerdo a la regla, que la transacción será abortada globalmente. Esto es conocido como la opción de aborto unilateral de los participantes.

Hay dos rondas de intercambio de mensajes entre el coordinador y los participantes: de allí el nombre de protocolo 2PC. Dos variantes importantes de 2PC son el aborto preasumido 2PC y el *commit* preasumido 2PC [Mohan *et al.*, 1986]. Son importantes porque reducen los mensajes y el costo de entrada/salida de los protocolos. El protocolo de aborto preasumido es incluido en el estándar XA X/Open, que ha sido adoptado como parte del estándar ISO para procesamiento distribuido abierto.

Una característica importante del protocolo 2PC es su naturaleza de bloqueo. Las fallas pueden ocurrir durante el proceso *commit*. La única manera de detectar estas fallas se efectúa por medio de un time-out del proceso esperando por el mensaje. Cuando esto ocurre, el proceso (coordinador o participante) que termina continúa un protocolo de terminación para determinar que hacer con la transacción que se encuentra en medio de la transacción *commit*.

Un protocolo *commit* de no bloqueo es aquel, cuyo protocolo de terminación puede determinar que hacer con una transacción en caso de fallas bajo cualquier circunstancia. En caso de 2PC, si una falla ocurre en el sitio del coordinador y un sitio participante mientras el coordinador esta colectando votos de los participantes, los participantes restantes no pueden determinar el destino de la transacción entre ellos y tienen que permanecer bloqueados hasta que el coordinador o el participante que falló se recupere. Durante este periodo, los bloqueos sostenidos por la transacción no pueden ser liberados, lo cual reduce la disponibilidad de la base de datos [Mohan *et al.*, 1986].

Lo inverso a terminación es recuperación. Cuando un sitio en falla se recupera de una falla, las acciones que debe de tomar para recuperar la base de datos del sitio a un estado consistente, es del dominio de los protocolos de recuperación distribuidos. Considerar el lado de recuperación del caso, en el cual el coordinador del sitio se recupera y el protocolo de recuperación tiene que determinar que hacer con las transacciones distribuidas cuya ejecución estaba coordinando [Skeen, 1981].

Protocolos de replicación

En bases de datos distribuidas replicadas[1], cada elemento de datos lógico tiene varios casos físicos. La cuestión en este tipo de sistemas de bases de datos es mantener la consistencia entre las copias. El criterio de consistencia más discutido es la equivalencia de una copia, la cual asegura que los valores de todas las copias de los elementos de datos lógicos deben ser idénticos cuando la transacción que actualiza termina.

Si la transparencia de replicación se mantiene, las transacciones en cuestión serán las operaciones de lecturas y escritura en un elemento lógico de datos *x*. El protocolo de control de réplica es responsable por el mapeo de operaciones sobre x a operaciones sobre copias físicas de *x*, (x_1, … x_n). Un típico protocolo de control de réplica que asegura la serializabilidad de una copia es conocida como protocolo leer una vez/ escribir todo (read-once/write-all, ROWA). Las transacciones de solo lectura pueden ser ejecutadas localmente, mientras que las transacciones de actualización son ejecutadas en todos los sitios en el sistema [Jiménez-Peris *et al.*, 2001].

Se han propuesto varios algoritmos alternativos para la replicación de bases de datos, los cuales difieren en el número de sitios contactados para realizar una operación de lectura o escritura.

2.3 Sistemas de archivos distribuidos

2.3.1 Introducción

En la actualidad existe un movimiento con tendencia a alejarse de las grandes computadoras *mainframe* para hacer uso de las estaciones de trabajo individuales. Sin embargo, un requerimiento fundamental para que se pueda dar dicha transformación es que sea posible compartir datos y programas, no solamente entre usuarios individuales en la misma computadora, sino que también a través de las máquinas en una red local o incluso en una red de área ampliada. Los sistemas distribuidos de archivos son programas de software que permiten el acceso a los datos y a otras

[1] La replicación no es un motivo de preocupación de SMBDs paralelas debido a que los datos no son normalmente replicados a través de procesadores múltiples. La replicación podría ocurrir como resultado del envío de datos durante la optimización de consultas, pero esto no es manejado por los protocolos de control de réplica.

computadoras en la red, de una forma transparente, por lo que se realiza el acceso a archivos distribuidos como si el acceso fuera a disco local. Los datos en un sistema de archivos distribuidos pueden residir sobre más de una computadora remota, y se espera que no solo los datos sean compartidos sino también su organización y atributos, tales como nombres, rutas, subdirectorios y derechos de acceso [Többicke, 1994].

El propósito de un sistema de archivos distribuidos (Distributed File System, DFS) es permitir a los usuarios de computadoras distribuidas físicamente, compartir datos y almacenar recursos usando un sistema de archivos común. Una configuración típica para una DFS es un conjunto de estaciones de trabajo y mainframes conectadas por una red de área local (LAN). Un DFS es implementado como parte del sistema operativo de cada una de las computadoras conectadas [Levy y Silberschartz, 1990].

Un sistema de archivos es un subsistema de un sistema operativo cuyo propósito es proporcionar almacenamiento de largo término; lo cual se realiza mediante la implementación de archivos, que son considerdos objetos nombrados que existen desde su creación explícita hasta su destrucción explícita y son inmunes a fallas temporales del sistema.

Un DFS es una implementación distribuida del modelo de tiempo compartido de un sistema de archivos, donde múltiples usuarios comparten archivos y recursos de alamacenamiento.

El propósito de un DFS es soportar la misma clase de compartición cuando los usuarios están físicamente dispersos en un sistema distribuido. Un sistema distribuido es un conjunto de máquinas débilmente acopladas – una mainframe o una estación de trabajo – interconectadas por una red de comunicación. A menos que se especifique de otra manera, la red, es una red de área local. Desde el punto de vista de una máquina especifica en un sistema distribuido, el resto de las máquinas y sus respectivos recursos son *remotos* y los recursos propios de la máquina son *locales*.

Algunos sistemas de archivos que han sido exitosos en el mundo de las computadoras personales son AppleShare para computadoras Macintosh y LAN Manager y Novell Netware para PCs compatibles con IBM. En el mundo Unix la referencia usual es Network File System (NFS) de Sun Microsystem. También Apollo Computer del dominio OS y Andrew File System (AFS) de Transarc son bastante populares. Otros sistemas distribuidos de archivos de Unix son RFS de AT&T, Sprite y sistemas experimentales como Amoeba [Többicke, 1994].

2.3.2 Tendencias y terminología

Idealmente, un DFS debería ver sus clientes como un sistema de archivos centralizado. Esto es, la multiplicidad y dispersión de los servidores y dispositivos de almacenamiento deberían ser transparentes para los clientes. La transparencia tiene muchas dimensiones y grados. Una propiedad fundamental, llamada *transparencia de red*, implica que los clientes deberían ser capaces de acceder archivos remotos usando el mismo conjunto de operaciones de archivo aplicables a los archivos locales. Esto es, la interfaz de cliente de un DFS no debería distinguir entre archivos locales y remotos [Levy y Silberschartz, 1990].

Otro aspecto de transparencia es la movilidad de usuario, la cual implica que los usuarios pueden acceder a cualquier máquina en el sistema; esto es, no están forzados a usar una máquina específica. Un DFS transparente facilita la movilidad del usuario, trayendo el ambiente del usuario (por ejemplo, su directorio de inicio) en cualquier lugar en donde ingrese al sistema.

La más importante medida de rendimiento de un DFS es la cantidad de tiempo requerida para satisfacer los requerimientos de servicio. En sistemas convencionales, este tiempo consiste del tiempo de acceso al disco y la cantidad de tiempo de procesamiento del CPU [Levy y Silberschartz, 1990]. En un DFS, un acceso remoto tiene un costo adicional atribuido a la estructura distribuida. Este costo incluye el tiempo requerido para entregar el requerimiento al servidor, así como también

el tiempo necesitado para que el cliente obtenga una respuesta de regreso a través de la red. Para cada dirección, además a la transferencia actual de la información, existe el costo de CPU de ejecutar el software de protocolo de comunicación. El rendimiento de un DFS puede ser visto como otra dimensión de su transparencia; esto es, el rendimiento de un DFS debería ser compatible al de un sistema de archivos convencional.

Se usa el término tolerancia a fallas en un sentido amplio: Fallas de comunicación, fallas de máquinas, fallas de dispositivos de almacenamiento y bajo rendimiento de los dispositivos de almacenamiento, todas ellas son consideradas como fallas que deberían ser toleradas en cierta medida.

Un sistema tolerante a fallas debería continuar funcionando, quizá de una forma degradada, cuando se presentan estas fallas. La degradación puede ser en desempeño, funcionalidad o en ambos, pero debería ser proporcional, en algún sentido, a la falla que lo causa. Un sistema que lleva a la suspensión cuando un pequeño número de sus componentes falla no es tolerante a fallas.

La capacidad de un sistema de adaptarse a la carga de servicio incrementado es llamada *escalabilidad*.

Los sistemas han delimitado recursos y pueden convertirse en completamente saturados bajo una carga incrementada. Con respecto al sistema de archivos, la saturación ocurre, por ejemplo, cuando un CPU de un servidor se esta siendo usado en una proporción muy alta o cuando los discos están casi llenos. Como para un DFS en particular, la saturación del servidor es incluso una gran amenaza del sobre-costo de comunicación asociado con el procesamiento de requerimientos remotos.

Aún un diseño perfecto no puede ajustarse a una carga de crecimiento constante. Adicionar nuevos recursos podría resolver el problema, pero podría generar carga indirecta adicional sobre otros recursos (por ejemplo, adicionar máquinas a un sistema distribuido puede saturar la red e incrementar las cargas de servicio). O aún peor, expandiendo el sistema se puede incurrir en modificaciones de diseño muy costosas. Un sistema escalable debería tener el potencial para crecer sin los problemas mencionados.

La tolerancia a fallos y escalabilidad están mutuamente relacionadas. Un componente altamente cargado se puede convertir en paralizado y comportarse como un componente en falla. También, cambiar una carga de un componente en falla a su respaldo puede saturar este último. Generalmente, tener recursos de soporte es esencialmente por confiabilidad, como también para manejar cargas pico de mejor manera [Levy y Silberschartz, 1990].

Una ventaja de los sistemas distribuidos sobre los sistemas centralizados es el potencial para tolerancia a fallas y escalabilidad debido a su multiplicidad de recursos. Un diseño inapropiado puede obstaculizar este potencial y aún peor puede hacerlo propenso a fallas. Tolerancia a fallas y consideraciones de escalabilidad deben tener un diseño basado en la distribución de control y datos. Cualquier entidad centralizada, ya sea un controlador central o un repositorio de datos centralizado, introduce un punto severo de falla o un cuello de botella en el rendimiento. Por lo tanto, un DFS escalable y tolerante a fallas debería tener servidores múltiples e independientes servidores controlando dispositivos múltiples e independientes de almacenamiento.

De hecho, que un DFS maneje un conjunto de dispositivos de almacenamiento dispersos es una característica distintiva clave. El espacio de almacenamiento total de un DFS consiste de diferentes espacios de almacenamiento, más pequeños, ubicados remotamente. Usualmente, hay una correspondencia entre estos espacios de almacenamiento y conjuntos de archivos. Se usa el término *unidad componente* para denotar el conjunto más pequeño de archivos que pueden ser almacenados en una sola máquina, independientemente de otras unidades. Todos los archivos pertenecientes a la misma unidad componente deben residir en la misma ubicación. En un sistema

distribuido la estructura dispersa y su actividad deben ser transparentes para los usuarios [Levy y Silberschartz, 1990].

2.3.3 Clientes y servidores de archivos

Para explicar la estructura de un DFS, se definirán los conceptos de servicio, servidor y cliente. Un servicio es una entidad de software ejecutándose en una o más máquinas y proporcionando un tipo particular de función a clientes que son desconocidos con anterioridad. Un servidor es un servicio de software ejecutándose sobre una sola máquina. Un cliente es un proceso que puede involucrar un servicio, usando un conjunto de operaciones que forman su interfaz cliente [Levy y Silberschartz, 1990].

Usando la tecnología de arriba, decimos que un sistema de archivos proporciona servicios de archivos a clientes. Una interfaz de cliente para un servicio de archivo esta formada por un conjunto de operaciones de archivo. Las operaciones más primitivas son *Crear* un archivo, *Borrar* un archivo, *Leer* de un archivo y *Escribir* a un archivo. El componente de hardware que un servidor de archivos controla es un conjunto de dispositivos de almacenamiento secundario sobre los cuales los archivos son almacenados y desde los cuales son recuperados de acuerdo a los requerimientos del cliente. Se dice que un servidor, o una máquina, almacenan un archivo, lo que significa que un archivo reside en uno de sus dispositivos anexos. Se hace referencia a un sistema de archivos de un una computadora uniprocesador, como un *sistema de archivos convencional*.

Un DFS es un sistema de archivos, cuyos clientes, servidores y dispositivos de almacenamiento se encuentran dispersos entre las máquinas de un sistema distribuido. En consecuencia, la actividad de servicio tiene que ser realizada a través de la red, y en vez, de un solo repositorio de datos centralizado hay múltiples e independientes dispositivos de almacenamiento. La configuración concreta e implementación de un DFS puede variar; hay configuraciones donde los servidores se ejecutan sobre máquinas dedicadas, así como configuraciones donde una máquina puede ser servidor y cliente a la vez. Un DFS puede ser implementado como parte de un sistema operativo distribuido o, alternativamente, por una capa de software, cuya tarea es manejar la comunicación entre sistemas operativos convencionales y sistemas de archivos. Las características distintivas de un DFS son la multiplicidad y autonomía de clientes y servidores en el sistema.

El término *cliente* es usado para significar un sistema que accede archivos localizados dentro del sistema de archivos distribuidos. El término *servidor de archivos* significa un sistema que ofrece datos para acceder desde otros sistemas.

2.3.4 Datos en caché

En una instalación de cientos y posiblemente miles de estaciones individuales, es importante evitar la re-lectura de los datos usados frecuentemente a través de la red. El sistema operativo Unix coloca bloques de disco en memoria caché. La consistencia de los datos en caché puede ser garantizada, aún cuando los procesos concurrentes sobre la misma máquina modifiquen el bloque: Un solo bloque se ubica (en el espacio de datos del kernel) de tal forma que todos los procesos en la misma máquina accedan el mismo bloque [Többicke, 1994].

En el estándar NFS, la misma estrategia aplica: La estación cliente maneja los bloques caché en el usual buffer caché de entrada/salida. Apollo Domain OS [Satyanarayanan, 1989] trabaja implementado paginación de objetos mapeados a memoria. Debido a que la memoria principal es un recurso de gran valor en la computadora, los datos en memoria caché son usualmente de vida corta. AFS y DFS introducen otro nivel de manejo de memoria caché para dirigir este problema. Los archivos son manejados en caché en fragmentos sobre el disco local de cada estación cliente. Cuando un proceso manejador de caché requiere información de caché de diferentes servidores, el requerimiento es transparente para el usuario y la consistencia es asegurada.

2.3.4.1 Consistencia de caché

Sobre un solo sistema, los datos en caché pueden conservar la consistencia con los datos en disco, porque la consistencia es controlada desde un solo punto. Sin embargo, las cosas cambian con los datos distribuidos, debido al envío de múltiples copias de fragmentos de datos involucrados.

Con los datos distribuidos, los cambios a los bloques sobre una estación de trabajo tienen que ser activamente propagados en otras estaciones. En los casos de un escritor y múltiples lectores, el resultado puede ser datos inconsistentes: Los sistemas de archivos distribuidos tienden a compartir archivos en fragmentos en vez de como un archivo completo y un programa secuencialmente leyendo un archivo podría comenzar con datos transferidos antes de que el archivo hubiera sido modificado y terminar con los datos que reflejan el estado después de la modificación.

Los sistemas de archivos distribuidos adoptan diferentes maneras para resolver este problema: Bajo Apollo Domain Os, un archivo no puede ser compartido simultáneamente entre lectores y escritores sobre diferentes máquinas. NFS simplemente ignora el problema. Normalmente se deja una ventana de 30 segundos en los cuales, los datos pueden ser inconsistentes.

AFS y DFS usan un mecanismo *callback*: El servidor de archivos conserva una pista de cuales clientes han manejado copias de caché y le indica al cliente invalidar su copia y que debería cambiar los datos. En AFS, el punto en el cual el servidor de archivos decide invalidar todas las copias pendientes es cuando el archivo es cerrado.

2.3.5 Seguridad

Los sistemas de archivos distribuidos han implementado mecanismos que refuerzan la seguridad del sistema; dichos mecanismos no garantizan la seguridad por si mismos; se deben seguir los procedimientos administrativos y operacionales adecuados a cada ambiente de trabajo [Satyanarayanan, 1990b].

Enseguida se describen dos de los mecanismos que han sido implementados para proporcionar seguridad del entorno que son la autenticación de usuarios y las listas de control de acceso.

2.3.5.1 Autenticación

Para conceder derechos de acceso a los datos, un servidor de archivos debe ser capaz de determinar la identidad del usuario que requiere el acceso. En *AppleShare* [APPLESHARE, 2007] y *Netware* [NETWARE, 2008], la identidad del usuario es establecida al inicio de la sesión usando un intercambio de password; así, el servidor de archivos establece la identidad del usuario.

En AFS y DFS, la autentificación de *Kerberos* es usada para establecer la identidad del usuario vía una tercera parte confiable. El proceso de autentificación no involucra passwords viajando sobre la red, en vez de esto, involucra la capacidad de desencriptar mensajes encriptados usando una llave derivada del password y publicación de respuestas significativas que prueban que el usuario conoce el password.

Un defecto tradicional de los servidores de archivos basados en NFS es que la ruta de un archivo determina la ubicación de un archivo. Cada sistema cliente monta partes de un espacio de archivos del servidor en un lugar en el sistema de archivos local. En el proceso se requiere, la dirección de red del servidor, y el cliente establece su propia vista de la topología total del servidor. Cambiar la ubicación de los datos de un servidor a otro requiere la propagación de esta información a todos

los clientes. Esta tarea puede ser aliviada a través de un proceso auxiliar (llamado automount); por lo cual sigue siendo importante el trabajo de configuración.

AFS y DFS introducen un concepto importante para grandes instalaciones: transparencia de ubicación. En AFS y DFS, los archivos son agrupados en unidades que residen en un solo servidor (en AFS estas unidades son volúmenes; en DFS son conjuntos de archivos). Cada unidad puede contener puntos de montaje en otras unidades sobre diferentes servidores. Un punto de montaje no contiene información acerca de la ubicación actual de los datos. En vez de esto, contiene una llave única asignada a través del dominio administrativo (referenciado como célula); esto es usado para buscar las direcciones de las unidades en una base de datos distribuida. Un efecto secundario conveniente del mecanismo de consulta es la emergencia de un espacio de nombres global.

En AFS y DFS, debido a que todos los clientes usan la misma base datos de consulta para incluir todos los puntos de montaje, todos interpretan la misma topología dentro de un mismo sistema de archivos. Debido a que toda la información de puntos de montaje esta contenida dentro del sistema de archivos, los cambios en la topología son visibles para todos los clientes.

2.3.5.2 Derechos de acceso

Mientras más datos sean accesibles a los usuarios, más compleja es la tarea de controlar el acceso a ellos. En NFS, el control de acceso se confía en la distinción estándar de propietario/grupo/otros (*owner/group/others*).

AFS ignora los bits de acceso estándar Unix *grupo* y *otros* y verifica los requerimientos de acceso contra una *lista de control de acceso* (ACL, por sus siglas en inglés). Las ACLs son definidas a nivel de un directorio Unix y se aplican a todos los objetos dentro del directorio. Los directorios hijo en turno tienen sus propias ACL, las cuales son inicializadas a la ACL del directorio padre. Una ACL controla los derechos de acceso al directorio mismo – los cuales pueden ser consulta (*lookup*), inserción (*insert*), borrado (delete) y administrador (*administer*, el derecho de cambiar la ACL) – y los objetos en el directorio – que son lectura (*read*), escritura (*write*) y bloqueo (*lock*). Los derechos de acceso son concedidos a un identificador de usuario de AFS, el cual es normalmente equivalente al usuario en Unix o a un miembro de un grupo de protección, donde un grupo de protección es un conjunto de IDs de usuarios de AFS. Sobre una ACL, se pueden especificar múltiples usuarios y grupos diferentes derechos.

2.3.6 Nombrado y transparencia

Nombrado es un mapeo entre los objetos lógicos y físicos. Los usuarios tratan con los objetos de datos lógicos representados por los nombres de archivo, mientras que el sistema manipula los bloques físicos de datos almacenados en las pistas de los discos. Usualmente, un usuario se refiere a un archivo por su nombre textual. El último es mapeado a un identificador numérico de más bajo nivel, el cual a su vez es mapeado a bloques de disco. Este mapeado multinivel provee a los usuarios con una abstracción de un archivo que esconde los detalles de cómo y donde se almacenan los archivos en el disco.

El concepto de archivos como abstracciones lleva a la noción de replicación de archivo; dado un nombre de archivo, el mapeo regresa un conjunto de ubicaciones de las réplicas de archivos correspondientes.

Hay dos nociones en relación al mapeo de nombres en un DFS:

Transparencia de ubicación: El nombre de un archivo no revela pista alguna de su ubicación de almacenamiento físico.

Independencia de ubicación: El nombre de un archivo requerido no será cambiado cuando la ubicación de almacenamiento físico cambie.

Existen 3 metodologías principales para esquemas de nombrado en un DFS. En el más simple de los métodos, los archivos son nombrados por alguna combinación de su *hostname* y el nombre local, el cual garantiza un nombre único en todo lo ancho del sistema.

El segundo método, popularizado por NFS de SUN, proporciona medios para que las máquinas individuales anexen directorios remotos a sus espacios de nombres locales. Una vez que el directorio remoto es anexado localmente, sus archivos pueden ser nombrados de una manera transparente.

El tercer método logra la integración total entre los sistemas de archivos componentes, una sola estructura de nombres global cruza todos los archivos del sistema. Consecuentemente, el mismo nombre de espacio es visible a todos los clientes.

2.3.7 Semánticas para compartir

Las semánticas para compartir son un criterio importante para evaluar cualquier sistema de archivos que permita a múltiples clientes compartir archivos. En particular, estas semánticas especificarían cuando las modificaciones de datos por un cliente son observables por clientes remotos.

Se debe notar que las aplicaciones que usan un sistema de archivos para almacenar datos y poseen restricciones sobre procesos concurrentes, para garantizar la consistencia semántica de los datos; por ejemplo, las aplicaciones de bases de datos, deberían usar medios especiales (por ejemplo, bloqueos) para este propósito, y no confiar en las semánticas de base para compartir proporcionadas por el sistema de archivos.

2.3.8 Métodos de Acceso Remoto

Considerar un proceso cliente que requiere acceder a un archivo remoto. Considerando que el servidor que almacena el archivo fue localizado por el esquema de nombrado, se debe iniciar una transferencia remota de datos para satisfacer el requerimiento del cliente. Hay dos métodos complementarios para manejar este tipo de transferencia.

Servicio Remoto. Los requerimientos de acceso se entregan al servidor. La máquina servidor ejecuta los accesos y los resultados son enviados de vuelta al cliente. Esto es una correspondencia directa entre accesos y tráfico hacia y desde el servidor. Los requerimientos de acceso son traducidos a mensajes para los servidores y las respuestas de los servidores son empaquetados como mensajes enviados a los clientes.

***Caching*.** Si los datos requeridos para satisfacer el requerimiento de acceso no están presentes localmente, se envía una copia de estos datos del servidor al cliente. Usualmente, la cantidad de datos traída es mucho más grande que los datos actualmente requeridos. Los accesos son ejecutados sobre la copia de caché en el lado del cliente. La idea es retener los bloques de disco accedidos en caché, de tal manera que los accesos repetidos a la misma información pueden ser manejados localmente, sin tráfico adicional en la red.

2.3.9 Replicación de archivos

La replicación de archivos es una redundancia útil para mejorar la disponibilidad. El requerimiento básico en un esquema de replicación es que las diferentes réplicas del mismo archivo residan en máquinas independientes a la falla. Esto es, la disponibilidad de una réplica no es afectada por la

disponibilidad del resto de las réplicas. Es deseable “esconder” los detalles de replicación a los usuarios. Es la tarea del esquema de nombrado mapear a un nombre de archivo replicado en una réplica particular.

El principal problema asociado con las réplicas es la actualización. Desde el punto de vista del usuario, las réplicas de un archivo denotan la misma entidad lógica; así, una actualización a cualquier réplica debe ser reflejada en todas las otras réplicas.

Otro punto importante a analizar es la consistencia de los datos. Si la consistencia no es de mayor importancia, puede ser sacrificada por disponibilidad y rendimiento. En muchos casos, la consistencia de los datos no puede ser comprometida, y el precio pagado por incrementar la disponibilidad por replicación es un complicado protocolo de propagación de actualizaciones.

2.3.10 Escalabilidad

La escalabilidad debería ser una consideración primaria en el diseño de sistemas distribuidos. Por *escala*, se puede referir al número de nodos en un sistema distribuido. Alternativamente, puede ser el número de los usuarios del sistema. Otra interpretación es que escala se refiere al número de estructuras organizacionales abarcadas por el sistema.

Escalar debe ser reconocido como un factor primario que tiene influencia en la arquitectura e implementación de los sistemas distribuidos. Desempeño, operabilidad y seguridad son consideraciones dominantes en el diseño de estos sistemas. Caching de cliente, transferencia de paquetes de datos, autenticación mutua basada en tokens y organización jerárquica del dominio de protección, han emergido como mecanismos que mejoran la escalabilidad. La separación de intereses hecha posible por la especialización funcional, ha también proporcionado valor en escalabilidad. La heterogeneidad es un subproducto importante de crecimiento, pero los mecanismos disponibles para adaptarlo son rudimentarios. La separación física de clientes y servidores resulta ser un requerimiento crítico para la escalabilidad.

2.4 Conclusiones

En este Capítulo se han presentado dos de las aplicaciones de los sistemas distribuidos, las bases de datos distribuidas y los sistemas de archivos distribuidos. Se describieron las características principales de los sistemas mencionados, así como también los procesos que cada sistema ejecuta para obtener una alta disponibilidad e integridad de los datos. En el caso de las bases de datos, la estrategia usada para crear alta disponibilidad es el particionamiento de datos y la integridad de la información es conservada siguiendo protocolos de concurrencia y protocolos de confiabilidad.

Para el caso de los sistemas de archivos distribuidos, la alta disponibilidad de los datos se logra mediante la replicación de archivos en distintos servidores. La integridad de la información debe ser guardada también por los métodos de replicación de archivos.

Se hace notar que los métodos para conservar la integridad y disponibilidad de la información se basan en protocolos de propagación de actualizaciones que afectan directamente el desempeño de las aplicaciones distribuidas.

Capítulo 3
Plataformas para la Implementación de una Base de Datos Distribuida

En el presente Capítulo se describen las características principales de los dos sistemas distribuidos que se utilizan en los diferentes Casos de Estudio de esta Tesis: MySQL Cluster y Coda.

3.1 Introducción

La base de datos distribuida y los casos de estudio que se presentan en esta Tesis están basados en dos aplicaciones de código libre; una de ellas es el sistema manejador de bases de datos distribuidas MySQL Cluster y la otra es el sistema de archivos distribuidos Coda; ambos diseñados y construidos para proporcionar al usuarios finales alta disponibilidad y confiabilidad en la información manejada. A continuación se presenta una descripción detallada de las características más importantes de estas dos plataformas a fin de dar un panorama general del funcionamiento interno de ambas aplicaciones y el impacto de este diseño en su desempeño final.

3.2 MySQL Cluster

MySQL [MYSQL, 2008] es uno de los sistemas manejadores de bases de datos de código abierto SQL más populares de mundo. MySQL es desarrollado, distribuido y soportado por MySQL AB [MySQL 5.1, 2007] que es una compañía comercial, fundada por los desarrolladores de MySQL. MySQL ha desarrollado MySQL Cluster [MySQL 5.1, 2007], el cual es una versión de MySQL de alta disponibilidad y alta redundancia, diseñada para trabajar un ambiente computacional distribuido en memoria. Usa el motor de almacenamiento NDB cluster para tener varios servidores MySQL colaborando en un cluster. MySQL cluster es soportado por algunas plataformas como Linux, Solaris, Mac OS X, HP-UX [4] y otros sistemas operativos al estilo de Unix bajo un número de tipos de hardware. MySQL Cluster tiene una arquitectura share-nothing, de tal forma que cada miembro del cluster tiene su propia memoria y disco y esta clase de estructura permite trabajar con hardware de bajo costo y con un mínimo de requerimientos específicos de hardware o software. Además, MySQL Cluster puede ser implementado fácilmente y proporcionar características de tolerancia a fallos, recuperación de nodos, replicación de datos síncrona y ni un solo punto de falla.

MySQL Cluster está diseñado para proporcionar 99.999% de disponibilidad debido a que no tiene un solo punto de falla y al esquema de replicación manejado.

La parte correspondiente a MySQL Cluster es configurada independientemente de la configuración de MySQL Server. En MySQL Cluster, cada parte del cluster es llamada *nodo*, en este contexto una máquina no es considerada un nodo, sino que el término nodo es usado para referenciar un proceso. Es posible ejecutar más de un proceso en una sola computadora que se encuentra dentro del cluster, la cual recibe el nombre de *cluster host*.

Existen tres tipos de nodos en el cluster:
Nodo de Administración (nodo MGM): Este nodo maneja la configuración del sistema y es usado para modificar la configuración del cluster, tiene otras funciones administrativas tales como iniciar o parar el funcionamiento de nodos, ejecutar respaldos, proporcionar el estado de los nodos, etc. En general, solamente se requiere un nodo de este tipo, pero puede existir más de uno. Debido a que este nodo maneja el archivo de configuración para todos los nodos, el administrador debe iniciarse antes que todos los demás.

El nodo de datos (nodo ndbd): Este tipo de nodo contiene los datos que pertenecen al cluster, los datos son replicados dentro de los nodos de datos. No es necesario tener más de una replica, pero la redundancia de datos es configurada con al menos dos réplicas.

Nodo SQL: Este es un servidor MySQL que usa el motor de almacenamiento NDB y puede acceder los datos del cluster a través de una interfaz tradicional MySQL. Puede haber varios nodos SQL en el cluster.

Debe haber al menos uno de cada uno de los nodos antes mencionados para crear la configuración de cluster mínima, pero es importante resaltar que un sistema productivo debe haber varios nodos de datos y SQL para proveer accesibilidad y alta disponibilidad.

3.2.1. Particionamiento de datos en MySQL Cluster

El particionamiento de datos es un concepto importante en MySQL Cluster ya que explica como el cluster divide los datos que son alimentados entre varios nodos de almacenamiento para lograr un alto desempeño y redundancia [Davies y Fisk, 2006]. Esto es, una tabla grande puede ser divida en varias tablas más pequeñas. Una función de partición es una función no-constante y no aleatoria (generalmente una función *hash*,) de uno o más campos en la tabla; en MySQL cluster, es siempre la llave primaria. Una función hash calcula un dato específico para un dato de entrada. No puede contener una consulta, pero si una expresión escalar. Actualmente, la función necesita regresar un entero. Además, la función debe ser relativamente simple ya que debe ser evaluada varias veces en consultas.

Se debe hacer notar que MySQL Cluster agrega automáticamente una llave primaria a cualquier tabla en donde no fue incluida, ya que se requiere una para el particionamiento.

El particionamiento es importante en los clusters, excepto en caso en que el número de replicas es igual al número de nodos de almacenamiento, porque de algún modo los datos deben ser divididos de igual forma en todos los grupos de nodos.

Para hacer la replicación de datos MySQL Cluster utiliza la replicación síncrona, lo cual significa que las consultas no se terminan hasta que los cambios han sido aplicados a todos los servidores involucrados. Lo anterior garantiza que todos los servidores tienen información consistente, pero el rendimiento puede representar un problema, aun cuando se elimina la operación de replica de los archivos de bitácora, requerida por las arquitecturas de disco compartido para un *failover* exitoso. Cada nodo dentro de un grupo de nodos contiene la misma información, por lo que mientras exista un nodo activo de cada grupo, el cluster puede seguir funcionando, ya que los datos se encuentran completos.

3.2.2. Rendimiento MySQL Cluster

MySQL Cluster tiene un impacto diferente en los conceptos de tiempo de respuesta, rendimiento y escalabilidad que el de un sistema aislado o una base de datos basada en disco, tal como las generadas con los motores MyISAM o InnoDB [Davies y Fisk, 2006].

En general el tiempo de respuesta con MySQL Cluster es mayor que el obtenido con un sistema normal. Si se considera la arquitectura de MySQL Cluster, esto toma mayor sentido. Cuando se ejecuta una consulta en un cluster, la consulta se realiza primero al servidor MySQL, y después se realiza en los nodos de datos y se envía la información de vuelta en el mismo sentido. Por lo anterior, resulta más rápido acceder recursos locales que leer los mismos a través de la red. Algunas consultas pueden ser más rápidas que otras debido al recorrido paralelo que es posible, pero no se puede esperar que todas las consultas tengan mejor tiempo de respuesta. Desempeño y escalabilidad son generalmente mucho mejores en clustering que una sola base de datos. En una base de datos de un sistema aislado, casi siempre se tiene un cuello de botella en entrada/salida a disco. Si se agregan usuarios y consultas, la entrada/salida a disco generalmente, se hace más lenta debido a un costo extra en la búsqueda.

Existen varios factores de los cuales depende el rendimiento de una base de datos distribuida, entre los que podemos señalar a la cantidad de memoria disponible, presentada con los *buffers* de I/O o con el espacio de trabajo para el procesamiento de consultas [Graefe y Shapiro, 1991]; el desempeño de la base de datos distribuida también se afectado por el caché de la base de datos y por los métodos de acceso a datos que son empleados [Elhardt y Bayer, 1984]. Si la misma consulta es ejecutada varias veces la memoria caché juega un papel importante ya que reduce el tiempo de respuesta de las solicitudes, de manera contraria, si las consultas realizadas son completamente diferentes, se reduce la efectividad de la caché [Davies y Fisk, 2006].

3.2.3. Índices

MySQL Cluster tiene tres tipos de índices: Índices de Llave primaria, índices hash únicos e índices ordenados (Árboles T) [Davies y Fisk, 2006].

Índices de llave primaria: Los datos en MySQL Cluster son particionados en base a un *hash* de la(s) columna(s) de llave primaria. Un índice *hash* puede ser utilizado para resolver consultas donde son usadas las igualdades, pero no puede ser usado para barrido de rangos. Es necesario usar toda la llave primaria, no solo una parte de ella.

Índices hash únicos: Los índices hash únicos se crean siempre que se declare la restricción UNIQUE en MySQL Cluster. Son implementados por medio de una tabla secundaria que tiene la columna declarada como UNIQUE y además, una llave primaria en la tabla base. Los índices UNIQUE pueden ser ligeramente más lentos que la llave primaria, pero aún así, son rápidos en la mayoría de las búsquedas.

Índices ordenados: Un índice ordenado en MySQL Cluster es implementado a través de una estructura llamada árbol T. un árbol T trabaja de la misma manera que un árbol B, pero está diseñado para uso de bases de datos en memoria. Las dos diferencias principales son que un árbol T es generalmente mucho más profundo que un árbol B (porque no tiene que preocuparse por las veces de búsqueda a disco) y que un árbol T no contiene los datos reales por si mismo, solo apunta a los datos, lo cual hace que un árbol T tome menos memoria que su similar árbol B.

3.2.4. Recuperación de datos en MySQL Cluster

MySQL Cluster tiene cuatro métodos diferentes para recuperar datos con diferentes características de desempeño: Acceso de llave primaria, acceso de llave única, acceso de índice ordenado y barrido de la tabla completa [Davies y Fisk, 2006].

Acceso de llave primaria: Cuando una consulta va a usar la llave primaria, está haciendo uso de una búsqueda normal de hash. El servidor MySQL forma un hash de la llave primaria, entonces, usando el mismo algoritmo para particionamiento, conoce exactamente cual nodo de datos contiene los datos y lo extrae de allí.

Acceso de llave única: Cuando una consulta va a usar una llave única, de nueva cuenta, se hace una búsqueda hash. Sin embargo en este caso, es un proceso de dos pasos. La consulta primero usa un hash sobre el valor UNICO para buscar el valor de la llave primaria que corresponde a la fila. Se hace esto usando el valor como la llave primaria dentro de una tabla escondida; entonces, cuando se tiene la llave primaria de la tabla base, ya es posible recuperar los datos del lugar apropiado. Para propósitos de tiempos de respuesta, el acceso UNICO es aproximadamente el doble del tiempo de respuesta de una búsqueda de llave primaria, porque tiene que hacer dos búsquedas de llave primaria para obtener una fila.

Acceso de índice ordenado: Para acceder un índice ordenado, el servidor MySQL hace lo que se conoce como barrido de índice paralelo (parallel index scan). Esto significa que se requiere a cada nodo de datos que busque a través del índice que el nodo de datos tiene localmente. Los nodos hacen esto en paralelo, y el servidor MySQL combina los resultados cuando son retornados al servidor desde los nodos de datos. Debido a la naturaleza paralela de este barrido, este tipo de índice, podría en teoría tener un mejor tiempo de respuesta que lo que se hace con una consulta local. Un barrido de índice ordenado causa más tráfico de red y consume más recursos que una búsqueda de llave primaria o de llave única. Sin embargo, puede resolver muchas consultas que no pueden ser resueltas usando una búsqueda de *hash*.

Barrido de la tabla completa: El método final para resolver una consulta es a través del uso del barrido de la tabla completa. MySQL Cluster puede hacer esto en dos formas diferentes, dependiendo de la versión y condiciones de inicio que se usaron en el servidor MySQL:

- *Barrido de la tabla completa sin la condición pushdown:* Esta primera opción es la más lenta y menos eficiente. En este método, todos los datos en la tabla son traídos al servidor MySQL, el cual entonces aplica una cláusula WHERE a los datos. Este es un método muy costoso, ya que al realizar una consulta en una tabla, todos los datos cruzan la red cada vez que se hace un barrido de tabla completa.
- *Barrido de la tabla completa con condición pushdown:* Hay una opción en MySQL, desde la versión 5.0 llamada engine_condition_pushdown. Si esta opción es activada, el servidor MySQL puede intentar hacer un barrido más optimizado de la tabla completa. En este caso la cláusula WHERE que se esta usando para filtrar los datos es enviada a cada uno de los nodos de datos. Entonces, cada uno de ellos puede aplicar dicha función antes de enviar los datos de vuelta por la red. Esto generalmente reduce la cantidad de datos que están siendo enviados y puede agilizar la consulta en relación a los métodos anteriores. Para activar esta condición es necesario modificar el archivo de configuración de MySQL server.

3.2.5. Alta disponibilidad en MySQL Cluster

MySQL Cluster tiene tres niveles de seguridad dirigidos a fallas. El primer nivel maneja fallas de procesos y computadoras en el cluster y es llamado Recuperación de Nodo (*Node Recovery*) [Ronström, 2005]. MySQL Cluster puede manejar fallas únicas en los nodos de datos del cluster, así como también ciertas situaciones de multi-falla. El segundo nivel maneja situaciones donde el cluster entero ha fallado, esto es llamado Recuperación del Sistema. En este caso el cluster se recupera del almacenamiento de datos en el sistema de archivos o de los nodos de datos.

El tercer nivel de seguridad es el respaldo en línea. Este puede manejar situaciones donde es necesario retroceder a una copia anterior de los datos cuando la aplicación causó inconsistencia de datos. Puede ser usada para ciertos procedimientos complejos de actualización no soportados en línea. Finalmente, puede ser usado como una tercera línea de seguridad cuando el cluster se paraliza y no puede retroceder por medio de un procedimiento en línea. La replicación MySQL puede ser usada en conjunto con MySQL Cluster [Ronström, 2005].

A continuación se presentan las razones detrás de las decisiones fundamentales de arquitectura tal como la elección de la arquitectura shared-nothing, los protocolos de *commit*, los protocolos checkpoint y otros componentes en la arquitectura de recuperación. En varios puntos se hace referencia a los nodos de datos en el cluster, por lo que en muchos casos nodos hace referencia a nodos de datos.

Arquitecturas de base de datos paralelas

En la literatura se han descrito varias arquitecturas para bases de datos paralelas. Estas son la memoria compartida, disco compartido y *shared-nothing* y hay también algunas hibridas. A

continuación, se hará una descripción de estas arquitecturas mencionadas y se dará una conclusión de porqué MySQL eligió la arquitectura *shared-nothing*.

Memoria Compartida

La memoria compartida es una arquitectura donde los procesadores comparten una memoria común y varios discos. Actualmente, esta arquitectura es muy común con los servidores multi-CPU como las computadoras Quad x86. Esta es una arquitectura que es soportada por todos los proveedores mayores de SMBDs. Los servidores con esta arquitectura son liberados por los mayores proveedores de hardware. La ventaja de una estructura de memoria compartida es que es fácil balancear la carga. En un SMBD, las implementaciones de un solo-procesador son fáciles de portar debido a que todos los datos son comunes como en un solo procesador. Tanto el paralelismo *inter-query* es como el intra-query pueden ser implementados en esta arquitectura.

Las arquitecturas de memoria compartida tienen varias desventajas: Una es que tiene una extensibilidad limitada; esto es debido a que todos los procesadores usan las mismas estructuras de memoria. Por lo tanto, algunos de los semáforos usados para serializar el acceso a memoria se convertirán en cuellos de botella cuando el número de CPU's se incrementen arriba de 20-40. Otro punto que necesita atención es como obtener tolerancia a fallas, ya que memoria y discos necesitan ser replicados. Sin embargo, el problema más difícil de resolver es obtener una buena confiabilidad, esto es, que un procesador ejecutándose con un error de software puede corromper datos en el sistema completo. Finalmente, los servidores con muchos CPU's usando una memoria compartida tienden a tener un precio mucho más alto por CPU debido a que los servidores comerciales actualmente tienen de 2 a 4 CPU's.

En conclusión, la arquitectura de memoria compartida es una manera fácil de extender los actuales SMBDs con paralelismo a una extensión limitada. Sin embargo, no es una buena solución a sistemas con altos requerimientos en tolerancia a fallas y desempeño escalable.

Disco compartido

El disco compartido es un método donde cada procesador tiene su propia memoria principal y esta no es accesible desde otros nodos. Sin embargo, los discos son comunes, así cualquier procesador puede acceder cualquier disco.

La mayor diferencia de disco compartido, comparado con memoria compartida, es que la memoria principal es privada, así que un procesador que falla no tiene la posibilidad de escribir en la memoria de otros procesadores. Esto resuelve el mayor problema de disponibilidad con la arquitectura de memoria compartida. Sin embargo, introduce un problema de coherencia con los buffers de disco. Debido a que no hay estructuras de memoria compartida, la arquitectura de discos compartidos escala mejor que una arquitectura de memoria compartida. Sus discos compartidos pueden, sin embargo, limitar la escalabilidad.

Para lograr una buena disponibilidad es esencial que los discos sean duplicados y posiblemente triplicados. La conclusión es que la arquitectura de discos compartidos puede manejar los requerimientos de tolerancia a fallas y en la mayoría de los casos también los requerimientos de desempeño. Tiene una ventaja en el balanceo de carga, pero tiene una complejidad más grande en la administración de buffers de disco. Es también complicado lograr cambios de software donde las estructuras en discos requieren cambios.

La última y más importante limitación en las arquitecturas de disco compartido es que el **failover** no es instantáneo. Debido a que los nodos en falla en el cluster han bloqueado recursos sobre el disco compartido, estos recursos deben ser desbloqueados y los registros de las bitácoras deben ser ejecutados antes que otros nodos puedan tomar estos recursos.

Shared-nothing

Shared-nothing significa que cualquier procesador puede acceder sus propios discos y su propia memoria y la comunicación entre sus nodos es solo a través del paso de mensajes. Esto significa que el sistema puede ser visto como un sistema de base de datos distribuido homogéneo. Así la investigación sobre sistemas de bases de datos distribuidas puede ser reutilizada para arquitecturas shared-nothing.

Debido a que las estructuras de datos no son compartidas, no hay **contención** de datos que obstaculizara la escalabilidad. Sin embargo, los datos pueden ser encontrados donde son replicados y así habrá contención para los recursos del procesador que limitan la escalabilidad. Sin embargo, se puede escalar a miles de procesadores usando el balanceo de carga y múltiples replicas de datos.

La disponibilidad es muy buena debido a que la falla de un nodo procesador no afecta ningún otro nodo procesador, solo se enviarían mensajes erróneos que deberían ser encontrados por verificación de error en la recepción de mensajes.

La arquitectura de *shared-nothing* proporciona muchas posibilidades de mejorar la disponibilidad. Todo esto requiere de soluciones de software. La arquitectura *shared-nothing* tiene muchas posibilidades de mejorar disponibilidad y escalabilidad.

En particular, *shared-nothing* proporciona la habilidad de manejar los requerimientos de *failover* en tiempos del orden de 1-3 segundos; esto no es posible con otra solución. También el desarrollo de mecanismos de comunicación eficientes hace el costo de paso de mensajes cada vez más pequeño. El desarrollo de mecanismos de comunicación más eficientes ha tenido un crecimiento más grande que el de CPU's y memorias. También la arquitectura *shared-nothing* encaja bien dentro de las soluciones de cluster con racks de servidores de mercado actuando juntos para lograr una arquitectura escalable DBMS.

La elección de arquitectura

MySQL Cluster tenía en sus metas de diseño original la habilidad de manejar failover en sub-segundos. Esto solo se logra con una arquitectura *shared-nothing*. Otro requerimiento en las metas de diseño original fue la habilidad de usar placas de CPU embebidas, que son populares en el sector de telecomunicaciones. La ventaja de esto ha crecido aún más en el tiempo, como el costo de los servidores con discos locales ha decrecido mucho más rápido que el costo de los subsistemas de disco compartido vendidos por la mayoría de los fabricantes de medios de almacenamiento.

Una ventaja de la arquitectura de disco compartido es que no hay necesidad de replicación entre los nodos en el cluster. Esto es verdadero, pero esencialmente esto significa que la solución de alta disponibilidad es liberada por el subsistema de disco en vez de por el SMBD [Ronström, 2005]. La solución de disco no compartido sería viable sin amplia replicación en un subsistema de disco. Subsistemas de disco que pueden manejar estos requerimientos tienen un costo muy alto, comparado con los servidores comerciales ejecutando el SMBD. Así, con MySQL Cluster el usuario obtiene una solución clusterizada completa. Solo es necesario comprar un conjunto de servidores comerciales equipados con memorias grandes, con un mecanismo de red estándar para conectar las computadoras. Usar una interconexión de cluster de alta velocidad sería benéfico, pero no es necesario. No es necesario hardware o software especial para lograr objetivos de clustering y auto-recuperación. Así MySQL Cluster está perfectamente en línea con las tendencias de desarrollo de cluster con cantidades masivas de servidores comerciales.

Protocolo de transacción

Una parte importante del manejo de la recuperación es el protocolo de compromiso de fase. Debe ser eficiente para lograr un alto rendimiento, debido a que es parte de todas las transacciones de escritura y para generar las condiciones que permitan que la base de datos distribuida proporcione alta disponibilidad, confiabilidad y un mantenimiento adecuado [Boutros y Desai, 1996]. Debe ser diseñado para evitar bloqueo en casi todas las situaciones; al mismo tiempo debe ser diseñado para manejar requerimientos de tiempo-real.

El algoritmo básico del protocolo de dos fases es el siguiente:

Durante la fase 1, inicialmente el coordinador envía un mensaje de consulta de commit a todos los participantes. Entonces espera que todos los participantes respondan al mensaje. En los participantes, si la transacción fue exitosa, se escribe una entrada en la bitácora UNDO log y una entrada en la REDO log. Entonces los participantes responden con un mensaje que manifiesta su acuerdo. Los participantes responderán con un mensaje ABORT, si la transacción falló.

Durante la fase 2, si el coordinador recibe un mensaje de acuerdo de todos los participantes, entonces escribe un registro COMMIT a sus bitácoras y envía un mensaje commit a todos los participantes. El coordinador espera por todos los mensajes ACK de los participantes. Si el participante recibe un mensaje commit, liberará todos los recursos y los bloqueos registrados durante la transacción y enviará un mensaje de acuse de recibo (ACK) al coordinador. Si el mensaje es ABORT, entonces el participante deshace la transacción con la bitácora UNDO log y libera los recursos y bloqueos requeridos durante la transacción, entonces envía un mensaje ACK.

En la Figura 3.1, se ilustra el envío de mensajes normal en la fase *commit*. Éste es el componente básico del protocolo compromiso de dos fases usado en MySQl Cluster. La mayor razón para el envío de mensajes *commit* en el orden opuesto es asegurar que los nodos de datos primarios reciben el último mensaje. Así el nodo de datos primario puede liberar los bloqueos en la fase *commit*. El primario siempre bloquea el primer registro y los nodos de respaldo no necesitarían bloquear ningún registro. Sin embargo, para habilitar lecturas del respaldo, los nodos de respaldo también bloquearán los datos. La razón para comenzar los mensajes de preparación en el nodo de datos primario es que *prepare* y *update* son puestos juntos. El mensaje *update* irá al nodo de datos primario debido a que este nodo recibe el requerimiento de bloqueo primero. El mensaje completo puede enviarse en cualquier orden; la razón para enviarlos en una forma lineal es decrementar el número de mensajes. La confirmación del *prepare*, el *commit* y los mensajes completos son enviados en señales empaquetadas especiales para bajar el costo de transacciones.

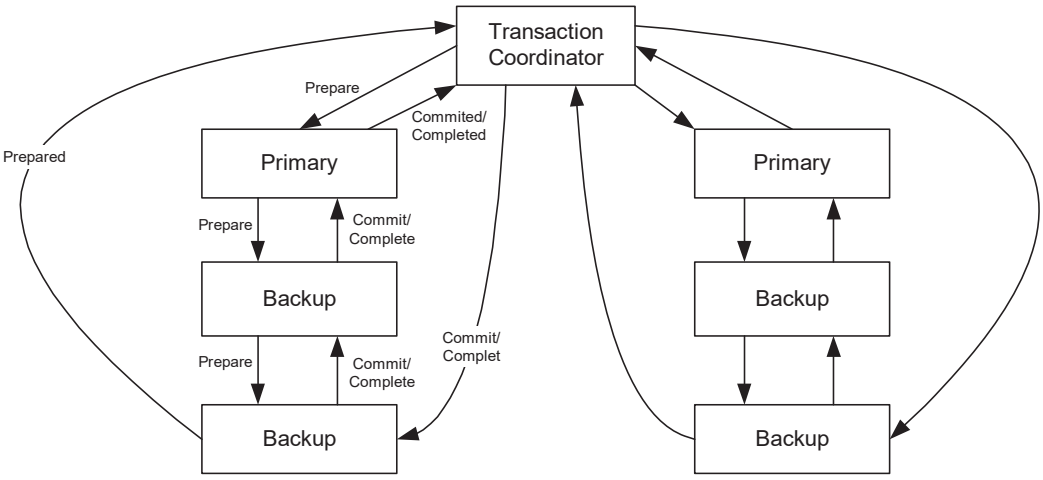


Fig. 3.1 Variante del protocolo de compromiso de dos fases. Ejemplo presentando una transacción con dos operaciones usando tres replicas.

Un estatuto SELECT a menudo lee la última versión comprometida (committed) y no se realiza espera debido al retardo por la liberación del bloqueo. Si los nodos de respaldo reciben una operación de escritura nueva sobre un objeto que esta bloqueado, comprometido (committed) pero no completado, entonces el respaldo puede liberar con seguridad el bloqueo del registro comprometido (committed) debido a que el primario, obviamente, ya ha liberado su bloqueo. Por lo que no es necesario una espera extra debido a transacciones incompletas.

Las escrituras a disco no son ejecutadas como parte del protocolo de compromiso de fase. La confiabilidad es obtenida por la escritura de la información *commit* en la memoria de todos los nodos participantes. Para asegurar la recuperación en situaciones de paro del cluster, un protocolo de *checkpoint* global asegura que cada transacción es escrita a disco de una manera consistente.

Protocolo de checkpoint global

Un *checkpoint* es un proceso por el cual MySQL Cluster escribe los datos y bitácoras en memoria a disco. Para el cluster, es un punto en tiempo donde todas las transacciones que han hecho un commit se guardan en disco. Hay dos tipos de checkpoint: local y global. En el checkpoint local, todos los datos que se encuentran en la memoria se escriben a disco. En el proceso de checkpoint global, la bitácora de REDO log se escribe a disco [Davies y Fisk, 2006].

Todas las transacciones y los registros de bitácora son marcados con la identidad del *checkpoint* global al que pertenecen. En la Figura 3.2 se muestra que los archivos REDO log son vaciados a disco durante un checkpoint global y las transacciones T1 y T2 se vuelven persistentes a disco. Ésto puede ser usado en situaciones de reinicio, para recuperación de fallas totales. El maestro está coordinando este protocolo. El tiempo de retraso límite alcanzado en la ejecución del protocolo en la más alta prioridad se conserva por debajo de 0.5-1.0 ms., en la mayoría de los casos. Cuando se recibe un mensaje *prepare*, todo el procesamiento *commit* en la transacción coordinador se suspende y el *checkpoint* global ID se actualiza con el valor nuevo. Cuando los mensajes *activate* llegan, el proceso *commit* puede comenzar otra vez y todos los *commits* subsecuentes usan el nuevo global *checkpoint* ID.

Solamente el procesamiento *commit* en la transacción coordinador se detiene. Las operaciones ejecutadas antes de alcanzar el punto de *commit* en la transacción coordinadora puede continuar y también las operaciones que han pasado el punto *commit* en la transacción coordinador. Los cambios en la información del esquema necesitan ser parte de esta transacción consistente *checkpoint*; esta es una de las razones de porque las operaciones esquema en MySQL Cluster son lentas, en comparación a bases de datos no distribuidas.

Cuando un *checkpoint* global ha iniciado, el *checkpoint* global anterior debe completarse y vaciarse a disco. Cuando todas las transacciones que pertenecen a este *checkpoint* global han concluido, se han vaciado a disco y el sistema de archivos del sistema ha sido actualizado, el *checkpoint* global es recuperable y puede crearse un nuevo *checkpoint*.

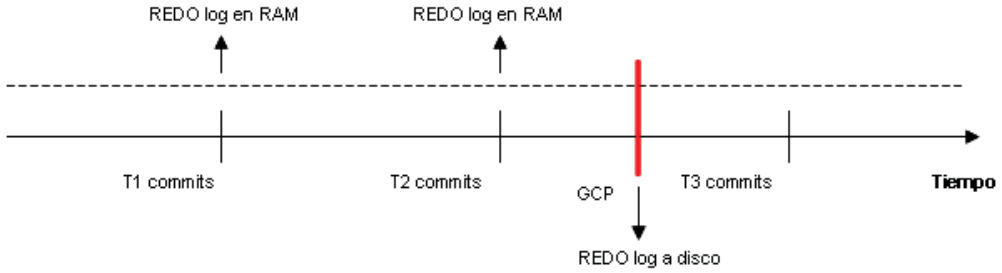


Fig. 3.2 Sistema de recuperación, checkpoint global. Los archivos REDO log son descargados a disco.

El parámetro de configuración *TimeBetweenGlobalCheckpoints* configura el tiempo antes de comenzar el siguiente *checkpoint* global.

Protocolo checkpoint local

Durante un checkpoint local (LCP) todos los datos en el cluster son almacenados en disco como se muestra en la Figura 3.3. Ésto también ocasiona un corte en la secuencia de la bitácora REDO log.

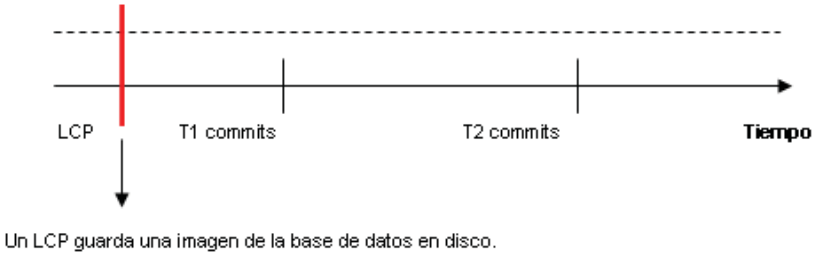


Fig. 3.3 Sistema de recuperación, *checkpoint* local.

Si el cluster falla, entonces es necesario reestablecer los fragmentos de todas las tablas usando la información guardada en almacenamiento permanente (por ejemplo, discos). Para que se pueda ejecutar un reinicio del sistema, es necesario almacenar información durante el procesamiento normal de la transacción. Un elemento necesario para reiniciar el sistema es el REDO *log*.

Esta bitácora (REDO *log*) almacena todas las operaciones de los fragmentos, así como también la información de la salida de la transacción. Esta bitácora es lógica debido a que es necesario una ubicación y replicación independiente. Una bitácora lógica solamente almacena información acerca de las operaciones de la base de datos y no información sobre las estructuras de datos internas. Esencialmente, una bitácora lógica contiene estatutos como UPDATE TABLE A=1, B=2 WHERE PrimeryKey=2, codificados en formato binario compacto. Esta independencia hace posible que una bitácora producida en un nodo pueda ser reestablecida en otro. Así, un nodo puede ser reestablecido usando un *checkpoint* local y un REDO *log* que residen en otro nodo.

Debido a que una bitácora guarda la información lógica de las operaciones, estos registros de bitácora pueden ser usados solamente en una estructura de datos donde las referencias entre índices y registros, y los registros dentro, sean consistentes. Así, antes de que los REDO *logs* puedan ser aplicados, debe crearse una acción consistente de la base de datos. Esto significa que la base de datos refleja un estado donde no hay acciones activas; las transacciones pueden, sin embargo, estar vivas. MySQL Cluster no reestablece un *checkpoint* con transacciones no vivas. Esto es llamado transacción consistente y también es una acción consistente.

Un método para lograr esto es deteniendo la base de datos mientras se esta creando una transacción consistente de *checkpoint*. Así, los requerimientos de tiempo real no serían reunidos; de ahí que esta solución no sea usada. En vez de esto se utiliza, un método difuso de *checkpoint*. Los *checkpoints* difusos no producen transacciones consistentes de *checkpoints*. De ahí que hay una necesidad de agregar otra bitácora, que es usada junto con el *checkpoint* difuso para producir una transacción consistente *checkpoint*. Se seleccionó un UNDO *log* para esta tarea; a través de este, se restaura el estado al inicio del *checkpoint* local. Esta bitácora hace posible poner la estructura de datos en un estado consistente de transacción. Es una bitácora de acciones sobre las estructuras de datos internas de la base de datos. La base de datos es organizada en páginas y la bitácora guarda la acción requerida para deshacer cualquier acción en estas páginas. La producción y uso del checkpoint difuso y el UNDO *log* es local al nodo. Los registros UNDO *log* de una página son solamente producidos entre el inicio de un *checkpoint* local y la escritura del *checkpoint* local, como se muestra en la Figura 3.4.

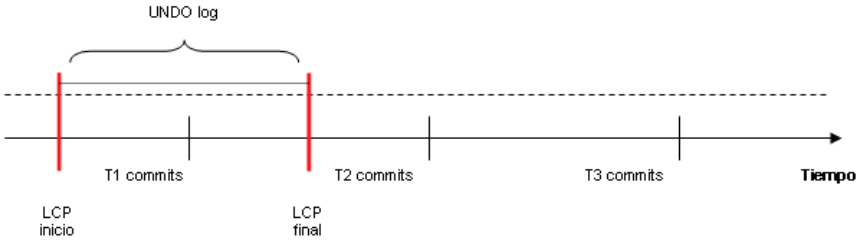


Fig. 3.4 Sistema de recuperación, UNDO log

Recuperación del nodo

A continuación se describirán los protocolos que manejan la recuperación después de fallas de nodos cuando las replicas primarias o de respaldo aún permanecen operacionales. Para soportar estos protocolos son necesarios varios protocolos básicos.

La recuperación del nodo consiste de varias actividades: La primera actividad es detectar nodos en fallas. Hay dos métodos de detectar un nodo en falla en MySQL Cluster. La primera es una conexión cerrada; esto usualmente pasa cuando el proceso se cierra de manera correcta y el sistema operativo cierra la conexión del proceso en falla. La segunda es a través de un mecanismo *heartbeat*; este mecanismo es usado siempre que un proceso falla de una manera silenciosa, por ejemplo, cuando la energía que alimenta a una computadora falla o si la conexión a la red es interrumpida.

La segunda actividad es manejar la falla del nodo. Esto involucra abortar transacciones relacionadas con el nodo que presenta la falla, excluir el nodo que falla del cluster de una manera controlada, manejar todos los protocolos en curso donde el nodo en falla estaba participando. MySQL Cluster implementa más de diez diferentes protocolos distribuidos y la mayoría de estos pueden manejar fallas de cualquier participante y fallas en el maestro. La mayoría de estos protocolos tienen un protocolo que asume un maestro (Master Take Over Protocol) y estos protocolos pueden manejar fallas de un nuevo maestro y de los participantes. Sin embargo, hay restricciones a la cantidad de fallas de nodo permitidas en un cluster para asegurar que el cluster no esta participando en dos clusters, y también que todos los datos en el cluster están siempre accesibles.

La tercera actividad es manejar el reinicio de un nodo en falla y sincronizar el inicio del nodo de datos con el cluster.

Manejo del protocolo heartbeat

La razón de tener un protocolo *heartbeat* es averiguar de una manera rápida, que nodos con errores son encontrados y que el resto de los nodos tengan un consenso de cuales nodos están activos y trabajando correctamente. Para ser mantenido en el conjunto confiable, cada nodo debe estar dentro de un cierto reporte de intervalo de tiempo a su vecino derecho que esta “vivo”. La idea de este protocolo es arreglar los nodos en una lista simplemente ligada. Cada nodo envía un h*eartbeat* a su vecino derecho en la lista ligada, como se muestra en la Figura 3.5.

Cuando un nodo no recibe ningún mensaje *heartbeat* en un número predefinido de intervalos (=4), el nodo envía un *NodeFailReport* al nodo maestro (el nodo más antiguo en el cluster). Si el nodo en falla es el maestro, se elige el nodo sobreviviente más antiguo en el cluster como el nuevo maestro y se envía el *NodeFailReport* a este último nodo. Este mensaje también actúa como una indicación para el receptor de que ha sido apuntado como el nuevo maestro.

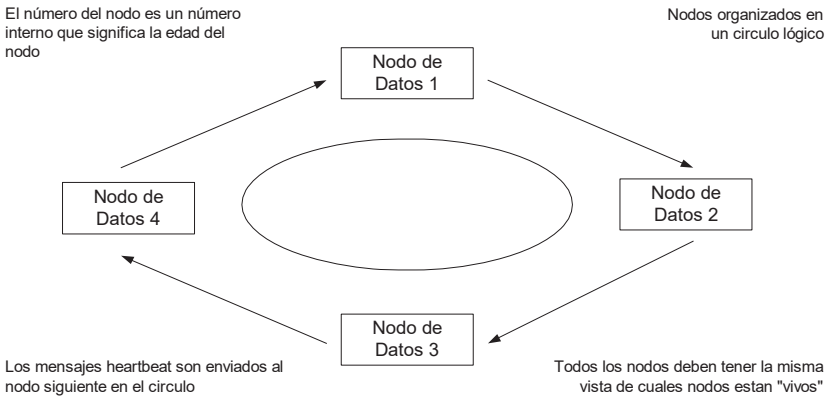


Fig. 3.5 Heartbeats

El nodo que descubrió el nodo en falla conserva esta información hasta que una nueva decisión ha sido activada donde el nodo en falla ha sido dado de baja. De esta forma, las fallas subsecuentes de maestros y de cualquier nodo, no llevarán a un conjunto activo incorrecto. Si un maestro propone un conjunto de nodos activos donde el nodo en falla es aún activo, el nodo reportará esto al maestro y no reconocerá el cambio propuesto. El maestro debe también remover el nodo en falla o el nodo reportando el nodo en falla.

El intervalo de tiempo entre el envío de *heartbeats* y la respuesta esperada es colocado por el parámetro de configuración *HeartBeatIntervalDbDb*.

Manejo de la transacción coordinadora

Durante el barrido de la tabla de transacciones participantes, cada transacción participante es verificada para revisar si la transacción coordinadora ha fallado. Si la transacción coordinadora de esta parte ha fallado, el nodo envía un reporte al Maestro. Este reporte contiene el estatus de la transacción, identifica el fragmento involucrado, la referencia al servidor MySQL y cual nodo fue involucrado en esta transacción. Cuando el proceso de barrido concluye, el nodo envía un mensaje de barrido completo al Maestro. Cuando el Maestro ha recibido toda la información de la transacción y ha recibido el barrido completo de todos los nodos “vivos”, entonces se tiene toda la información necesaria para tomar decisiones sobre las transacciones, con una transacción coordinadora en falla.

La decisión será abortar una transacción si ningún nodo reporta que ha escuchado una decisión *commit*. Si cualquier nodo ha reportado una decisión *commit*, se decide realizar un *commit* a la transacción. Si el Maestro también falla antes de que haya terminado su trabajo, otro coordinador es elegido automáticamente y este nuevo coordinador ejecuta las mismas acciones. De esta manera el método continúa hasta que todos los nodos hayan fallado o hasta que el sistema se encuentre a sí mismo en un estado inconsistente y decida ejecutar un paro del sistema. Si múltiples nodos fallan al mismo tiempo; el manejo de la falla de nodo será serializado.

Finalmente, el tiempo para manejar las transacciones fallidas, toma normalmente 1-2 segundos en un sistema cargado.

Particionamiento de la red y falla del cluster completo

Antes de proceder con el manejo de la falla del nodo, se debe asegurar que el cluster pueda manejar la falla del nodo. Hay dos riesgos, el primero es que el fragmento no tenga una replica

"viva", y en este caso el cluster no puede operar debido a que parte de los datos no son accesibles. El segundo es que el cluster es dividido en dos mitades, donde ambas mitades son operacionales.

Todas las replicas de los fragmentos pérdidas

El cluster no sobrevivirá si no todos los fragmentos tienen todavía al menos una réplica "viva". Hay dos posibles esquemas en la colocación de replicas de fragmentos. El primer esquema las extiende, de tal manera que copiar a un nodo de partida pueda ser eficiente. Esto significa que el sistema con 2 replicas (una primaria y un respaldo) puede manejar solamente una falla.

La otra opción es colocar todas las replicas de fragmentos en un grupo de nodos. En este caso el cluster puede sobrevivir mientras al menos un nodo por grupo de nodos este vivo. Sin embargo, al reinicio del nodo, solo un nodo esta disponible para restaurar los datos al nodo de inicio (en el caso de dos replicas).

MySQL ha implementado la ubicación de replicas de fragmentos en grupos de nodos como el la Figura 3.6. Así, detectar si el cluster puede sobrevivir es simplemente una manera de asegurar que al menos un nodo por grupo de nodos este "vivo". Esto tiene el beneficio de que MySQL puede sobrevivir a la falla de múltiples nodos.

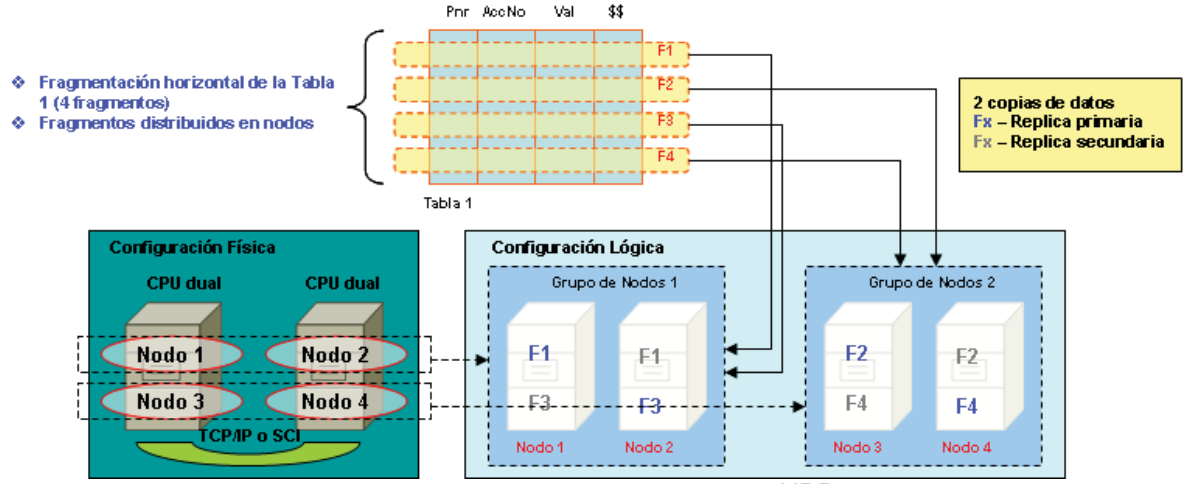


Fig. 3.6 Distribución de los datos en un cluster NDB

Uno de los beneficios de esto es que varios nodos de datos de diferentes grupos pueden ser colocados en la misma computadora física. El grupo de un nodo es elegido por su lugar en el archivo de configuración. Si hay dos replicas en el cluster, entonces el número de nodos de datos debe ser múltiplo de dos. Los primeros dos nodos forman el primer grupo de nodos, los siguientes dos forman el siguiente grupo de nodos y así sucesivamente.

Particionamiento de la red

Cuando todos los nodos de un grupo de nodos están vivos, no es posible obtener particionamiento de red. En este caso, la mitad del cluster no puede formar un cluster funcional.

El problema viene cuando ningún grupo de nodos tiene todos sus nodos vivos. En este caso un particionamiento de red es posible; se contacta un árbitro. Los nodos en el cluster han seleccionado el árbitro antes de la falla de nodo y todos los nodos contactarán al mismo árbitro. Normalmente, el árbitro sería el administrador del servidor, pero puede también ser configurado para ser cualquiera de los servidores MySQL en el cluster. El árbitro solamente aceptará la primera mitad del cluster en contacto para sobrevivir, a la segunda se le dirá que muera. La selección de árbitro es controlada por el parámetro de configuración ArbitrationRank en los nodos servidores MySQL y el servidor administrador.

Manejo fallas de un servidor MySQL

Cuando un servidor MySQL falla, cada nodo de datos lo detecta independientemente de los otros nodos de datos. Cada nodo de datos se asegurará de que cualquier transacción u operación en curso sobre el esquema de la base de datos en el servidor MySQL en falla se completa inmediatamente y de que los recursos usados en el nodo de datos son liberados.

Reinicio del nodo para un nodo de datos

Cuando una falla de nodo ha concluido, entonces el cluster esta listo para aceptar al nodo para su inclusión nuevamente, tan pronto como el nodo se inicie otra vez. El proceso del nodo de datos (ndbd) tiene un proceso el cual observa el nodo de datos real por falla y si el parámetro de configuración *StopOnError* es puesto a 1 ó *Y*; el proceso no se detendrá durante la falla del nodo, pero se reiniciará inmediatamente otra vez. Si se concluyen estos dos procesos, es necesario reiniciar el proceso y si la computadora o el sistema operativo falla o es reiniciado, entonces es necesario insertar una entrada en el arranque lógico del sistema operativo que reinicie el nodo de datos otra vez de manera automática durante el arranque de la computadora.

La lógica para reiniciar un nodo contiene seis fases importantes:

1. Inclusión del nodo en el cluster (para propósitos de *heartbeats* y manejo del cluster).
2. Copiado de Información del Diccionario.
3. Copiado de Información de Distribución.
4. Inclusión del nodo en el *checkpoint* global, *checkpoint* local y otros protocolos importantes.
5. Copiado de registros de datos al nodo de inicio.
6. Ejecutar un *checkpoint* local completo donde el nodo de datos de inicio esta participando.

Inclusión del nodo en el cluster

En las fases de inicio es requerido incluir el nodo dentro del cluster, de tal manera que el mecanismo *heartbeat* esta trabajando para el nodo que esta en la etapa de inicio.

Copiado de Información del Diccionario y de Distribución

Antes de que el nuevo nodo pueda ser parte de los protocolos distribuidos, necesita la información del esquema en el diccionario y la información de distribución. Esto es manejado bloqueando el diccionario y la información de distribución. El diccionario es solamente actualizado por las acciones del usuario, así que esto significa que CREATE TABLE y ALTER TABLE son temporalmente suspendidas durante un reinicio de nodo. El bloqueo de la información de distribución es ejecutado a través del paro del protocolo de checkpoint local. La actualización de la información de distribución ocurre como parte de la información del *checkpoint* local y cuando se ejecutan CREATE TABLE y ALTER TABLE. Cuando estos bloqueos son ejecutados, se realiza el copiado; es suficiente bloquear el maestro debido a que el maestro bloquea los cambios en el esquema y los protocolos de *checkpoint* local. La información de la distribución contiene información acerca de los fragmentos de la tabla, todas las replicas de los fragmentos, en cual nodo están almacenados y para cada replica del fragmento hay información acerca de los *checkpoints* locales que ha completado y que registros de la bitácora del *checkpoint* global son necesarios para restaurarse desde este *checkpoint* local.

Durante un reinicio de nodo, el nodo de inicio confía en el Maestro y en los otros nodos en el mismo grupo de nodos. Si cualquiera de estos nodos falla a la mitad del reinicio del nodo, también el nodo de inicio fallará.

Inclusión del nodo en los protocolos distribuidos

Cuando todos los metadatos han sido copiados al nodo de inicio, es tiempo de ser incluido en el protocolo de *checkpoint* global y en el protocolo de *checkpoint* local. Esto puede solamente pasar al inicio del *checkpoint* global, pero para los otros protocolos puede hacerse inmediatamente, debido a que un bloqueo previo ha suspendido los otros protocolos. Cuando esta inclusión ha sido ejecutada, los bloqueos pueden ser liberados, habilitando así las operaciones de esquema en MySQL Cluster otra vez. El tiempo de este bloqueo es normalmente de unos pocos segundos.

Copiado de los registros de datos al nodo de inicio

Una nueva replica del fragmento es creada a través del nodo maestro después de ejecutar un protocolo de transición de estado donde el cluster es informado de la copia nueva. La idea básica del protocolo de copia de fragmento es simple: El nodo primario envía el fragmento, registro por registro hasta que el último registro es enviado. Entonces se envía un mensaje al nodo de inicio informándole que el fragmento esta actualizado. El nodo receptor recibe todas las transacciones de escritura. Si el registro afectado ha llegado, se le aplica la transacción de escritura a la tupla. Si la tupla no ha llegado se ignora la escritura, pero participa en todas las fases del protocolo de compromiso de dos fases. Las operaciones de inserción son siempre procesadas en la nueva copia. Cuando el nodo de inicio escucha que el fragmento se ha copiado completamente, entonces el estado de cambio cierra la puerta para permitir actualizaciones sobre registros no existentes. Durante la copia de un registro, el registro es bloqueado para lectura; este bloqueo es muy corto. Esta optimización es posible debido a que se sabe que los mensajes entre el nodo primario y el nodo de inicio siempre llegan en orden. En un sistema cargado, el proceso de copia se alentará; de tal forma que las operaciones normales reciben una proporción más alta de la carga.

La mayoría del tiempo de recuperación de un nodo es gastado en la copia de fragmentos. Así que le tiempo de reinicio del nodo es mayormente dependiente de la cantidad de datos en el nodo. En un nodo cargado de datos normalmente podrá copiar unos MBytes de datos por segundo. De tal manera que el reinicio de un nodo puede tomar varios minutos en sistemas grandes.

Ejecutar un checkpoint local completo donde el nodo está participando

Hay dos pasos para declarar una nueva replica actualizada. El primer paso es reportar cuando el proceso de copiado esta terminando y la nueva copia contiene todos los registros. Después de este paso, el nodo puede ser usado para lectura. Antes de que la nueva replica pueda ser declarada como recuperable, el nodo debe ejecutar un checkpoint local. Esto significa que si el nodo falla después de declararse a sí mismo como actualizado, pero no recuperable todavía, no es útil en una situación de falla total. La razón es que solamente después del primer *checkpoint* local, la replica es almacenada en forma estable en disco. Este comportamiento es logrado por ejecutar un *checkpoint* local, después de que todos los fragmentos han sido copiados y antes de que el nodo sea declarado como iniciado.

Recuperación de Sistema

La recuperación del sistema es manejada por la preparación para un reinicio del sistema usando *checkpoints* locales, *checkpoints* globales y todos las bitácoras en el sistema. Entonces, si es necesario ejecutando un reinicio del sistema, el cual debería de ser un evento poco común (para alcanzar los cinco nueves de disponibilidad esto no debe de suceder más de una vez al año).

Un reinicio del sistema se realiza, primero, asignando el *checkpoint* global a restaurar. El esquema de información es restaurado para encontrar cuales tablas deberán ser recuperadas. La información de distribución es restaurada para encontrar todos los fragmentos de las tablas a restaurar e información de recuperación para cada fragmento. Entonces cada fragmento es recuperado y el sistema es preparado para iniciar cuando la información de distribución ha sido construida para el reinicio del sistema.

3.3 Coda

3.3.1 Introducción

Coda es un sistema de archivos para un ambiente computacional distribuido de estaciones de trabajo Unix que proporciona resistencia a fallas al servidor y a la red a través del uso de dos mecanismos diferentes, pero complementarios. Un mecanismo es la replicación de servidor, el cual almacena copias de un archivo en múltiples servidores. Y el otro mecanismo, la operación en modo desconectado, que es un modo de ejecución en el cual un sitio de *caching* asume temporalmente el rol de sitio de replicación. Este tipo de operación es principalmente usado por los equipos portátiles [Satyanarayanan *et al.*, 1990].

Coda es un sistema de archivos distribuido, que hace los archivos disponibles a un conjunto de computadoras cliente como parte de su árbol de directorios, manteniendo una copia de los archivos de datos en los servidores Coda [Braam *et al*., 2000].

El Sistema de Archivos Distribuidos Coda es un sistema de archivos experimental desarrollado en el grupo de M. Satyanarayanan en Carnegie Mellon University (CMU). Coda incorporó las características que otros sistemas no tienen [Braam P. J., 1998]:

- Tolerancia a fallas
- Desempeño y escalabilidad
- Seguridad
- Computación móvil
- Semánticas para compartir bien definidas
- Código fuente disponible libremente

Coda es un sistema de archivos distribuidos de ubicación transparente basado en un modelo de un sistema de archivos de Unix, dicho mecanismo permite la colaboración entre usuarios dispersos físicamente. Esto ha sido de gran utilidad en ambientes de estaciones de trabajo distribuidas donde las actividades principales son educación, investigación y desarrollo de software [Satyanarayanan, 1989].

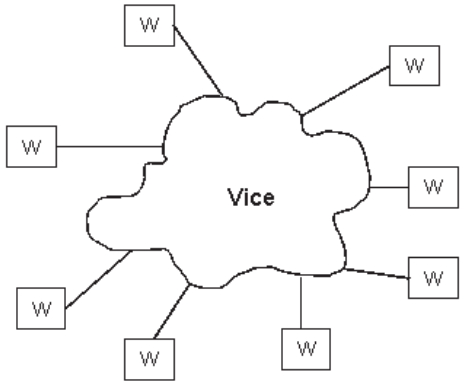
El sistema de archivos Coda es descendiente de *Andrew File System* (AFS) el cual es substancialmente más resistente a fallas. El ideal que Coda se esfuerza en alcanzar es la disponibilidad constante de los datos, permitiendo al usuario continuar trabajando a pesar de que se presentara alguna falla en alguna parte del sistema.

En los siguientes párrafos se describe la arquitectura de Coda, su esquema de seguridad y los elementos que han permitido que sea considerado como un sistema de archivos de alta disponibilidad.

3.3.2 Descripción general

El antecesor de Coda es Andrew File System el cual reúne una síntesis de las mejores características de computación personal y tiempo compartido [Satyanarayanan M., 1990b]. Una

vista conceptual de este modelo es presentada en la Figura 3.7. La estructura en el centro llamada Vice, es la información compartida en el sistema. Aunque representada como una sola entidad, consiste de un conjunto de servidores de archivos dedicados y una red de área local [Satyanarayanan M., 1990b].



Le estructura llamada "Vice" es una colección de servidores de confianza y una red de área local. Los nodos "W" son estaciones de trabajo públicas, que pueden acceder a la información de Vice. El software en cada nodo hace posible que los archivos compartidos en Vice aparezcan como parte integral del sistema de archivos de los nodos.

Fig. 3.7 Vista conceptual de la arquitectura de *Andrew*.

Compartir datos en Andrew es soportado por un sistema de archivos distribuido que aparenta ser un solo subárbol del sistema de archivos local en cada estación de trabajo. Los únicos archivos fuera del subárbol compartido son archivos temporales y archivos esenciales para la inicialización de la estación de trabajo. Un proceso llamado Venus, ejecutándose en la estación de trabajo, media el acceso a los archivos compartidos. Venus encuentra archivos en vice, los guarda en caché local y ejecuta la emulación de semánticas de sistemas de archivos Unix. Ambas Vice y Venus son invisibles a los procesos de la estación de trabajo, quien solo ve un sistema de archivos Unix, un subárbol que es idéntico en todas las estaciones de trabajo. Procesos de dos diferentes estaciones de trabajo pueden leer y escribir archivos en este subárbol como si estuvieran ejecutándose en un solo sistema de tiempo compartido.

3.3.3 Sistema de archivos distribuidos

Un sistema de archivos distribuido almacena archivos sobre una o más computadoras llamadas *servidores* y los hace accesibles a otras computadoras llamadas *clientes* y estos archivos aparecen como archivos normales, que pertenecen a la estructura local del cliente [Braam, 1998].

Existen varias ventajas en usar un sistema de archivos distribuido, como lo es que los archivos se encuentran más disponibles, debido al hecho de que muchas computadoras pueden acceder a un mismo servidor, además, compartir archivos desde una sola ubicación es más fácil que distribuir copias de archivos a clientes individuales. Otra ventaja es que los respaldos y la seguridad de la información son más fáciles de controlar debido a que solo los servidores deben ser respaldados. Los servidores pueden proporcionar una gran capacidad de almacenamiento, el cual podría ser costoso o impráctico si cada cliente proporcionara dicha capacidad.

Hay varios problemas que enfrenta el diseño de un buen sistema de archivos distribuido [Braam, 1998]:

- Transportar muchos archivos sobre la red puede crear un bajo rendimiento y latencia, dando como resultado cuellos de botella y sobrecarga del servidor.

- La seguridad de los datos es otro aspecto importante, ya que se debe tener la certeza de que un usuario esta autorizado para acceder la información.
- Otro problema son las fallas de la red, ya que estas, pueden dejar un cliente inaccesible o a un servidor inhabilitado para dar acceso a información importante.

3.3.4 Coda en un cliente

Si Coda se está ejecutando en un cliente, el cual es una estación de trabajo Unix, tecleando *mount* se desplegará el tipo de sistema de archivos "Coda" montados bajo /coda. Todos los archivos compartidos por cualquiera de los servidores están disponibles bajo este directorio, y todos los clientes acceden el mismo espacio de nombres. Un cliente se conecta a Coda y no a servidores individuales, los cuales son manejados de manera transparente. La idea del espacio de nombre global, no es nueva, el sistema de archivos Andrew [Satyanarayanan, 1989], el predecesor, pionero de la idea y almacena todos los archivos bajo /afs. Similarmente, el sistema de archivos distribuido DFS/DCE de OSF [Braam, 1998] monta sus archivos bajo un directorio directov.

Coda proporciona un solo punto de montaje, lo cual significa que todos los clientes pueden ser configurados de manera idéntica y que los usuarios siempre verán el mismo árbol de directorios. Para instalaciones grandes esto es esencial; en Coda un cliente necesita solamente conocer donde encontrar el directorio root Coda, /coda. Donde los nuevos servidores son agregados, el cliente descubrirá estos automáticamente en el árbol coda.

Cuando un programa intenta acceder a un archivo que pertenece al sistema Coda, dicho programa hará unas llamadas al sistema en relación al archivo. Una llamada al sistema es una operación a través de la cual el programa requiere al kernel por un servicio.

Cada cliente Coda debe instalar un módulo del núcleo que permite realizar la interfaz entre el Sistema Coda y el Sistema de Archivos Virtual de Linux (VFS, *Virtual File System*). Por ejemplo, cuando se abre un archivo, el kernel debe hacer una operación de búsqueda para encontrar el inode del archivo y regresa un manejador asociado con el archivo al programa. El inode contiene la información para acceder los datos en el archivo y es usado por el kernel; el manejador de archivo es para el programa que realizó la apertura del archivo. La llamada de apertura hace que el sistema de archivos virtual (Virtual File System, VFS) se active en el kernel y cuando es conocido que el requerimiento es por un archivo que pertenece al sistema de archivos /coda, dicho requerimiento es manejado por módulo del sistema de archivos Coda en el kernel. Coda es un módulo pequeño, conserva un caché de los requerimientos más recientes del VFS, pero de otra forma, pasa el requerimiento al manejador de caché Coda, llamado Venus. Venus verificará la caché de disco del cliente para el archivo requerido y en caso de que no exista en el caché, Venus contacta a los servidores Vice para obtener el archivo solicitado. Cuando el archivo ha sido localizado, Venus responde al kernel, el cual regresa la respuesta al programa solicitante como se muestra en la Figura 3.8.

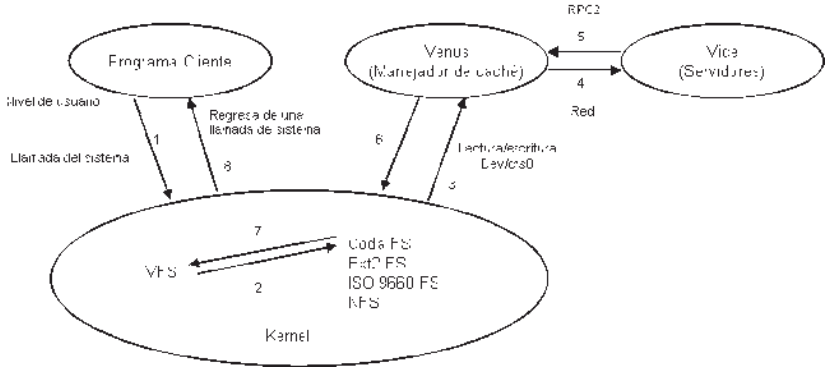


Fig. 3.8 Esquema de comunicación de Coda

La figura muestra como un programa de usuario requiere un servicio desde el kernel a través de una llamada del sistema. El kernel lo pasa a Venus, permitiendo a Venus leer el requerimiento desde el dispositivo de caracteres /dev/cfs0. Venus trata de responder el requerimiento, buscando en su caché, preguntando a los servidores o posiblemente declarando desconexión y sirviéndolo en modo desconectado. El modo desconectado entra en funcionamiento cuando no hay conexión de red. Típicamente esto ocurre para laptops durante las fallas de la red. Si los servidores fallan, la operación desconectada puede entrar en acción.

Cuando el kernel pasa el requerimiento *abrir* a Venus por primera vez, Venus trae el archivo entero desde el servidor, usando llamadas a procedimientos remotos para alcanzar los servidores. Entonces, almacena el archivo como un archivo contenedor en el área de caché. El archivo es ahora un archivo ordinario en el disco local, y las operaciones de lectura/escritura no se relacionan con Venus, sino que son manejadas casi totalmente por el sistema de archivos local (**ext2** para Linux). Las operaciones Coda lectura/escritura toman lugar a la misma velocidad que aquellas hacia archivos locales. Si el archivo es abierto por segunda vez, no será traído desde los servidores otra vez, sino que la copia local estará disponible para su uso inmediato. Los directorios (los cuales son archivos), así como todos los atributos (propietario, permisos y tamaño) son mantenidos en caché por Venus y Venus permite realizar operaciones sin contactar el servidor, si los archivos están presentes en caché. Si el archivo ha sido modificado y es cerrado, Venus actualiza los servidores enviando el archivo nuevo. Otras operaciones que modifican el sistema de archivos, también son propagadas a los servidores, tales como crear directorios, borrar archivos o directorios y crear o borrar ligas, etc. Así, Coda almacena en caché toda la información que necesita sobre el cliente y solo informa al servidor de actualizaciones hechas al sistema de archivos.

3.3.5 Volúmenes, servidores y replicación de servidores

Los archivos en servidores Coda no son almacenados en sistemas de archivos tradicionales. Las particiones en las estaciones de trabajo servidores de Coda pueden estar disponibles para el servidor de archivos. Estas particiones contendrán archivos que son agrupados en volúmenes. Cada volumen tiene una estructura de directorio como un sistema de archivos, por ejemplo, un directorio root para el volumen y un árbol de directorios debajo de este.

Coda conserva volúmenes y directorios de información, listas de control de acceso e información de atributos de archivos en particiones raw. Estas son accedidas a través de una memorial virtual recuperable basada en una bitácora (*Recovable Virtual Memory, RVM*) por velocidad y consistencia. RVM tiene soporte *build-in* para transacciones, esto significa que en caso de que un servidor tenga una falla, el sistema puede ser restaurado a un estado consistente sin mucho esfuerzo. Un volumen tiene un nombre y un ID y es posible montar un volumen en cualquier lugar bajo /coda.

Coda no permite puntos de montaje sobre directorios existentes, en vez de esto, se creará un nuevo directorio como parte del proceso de montaje. Esto elimina la confusión que puede surgir al montar sistemas de archivos Unix sobre directorios existentes.

Coda identifica un archivo por 3 enteros de 32 bits llamados un Fid: Consiste de un VolumeId, un VnodeId y un Uniquifier. El VolumeId identifica el volumen en el que el archivo reside. El VnodeId es el número "*inode*" del archivo y los *uniquifiers* se requieren para resolución. El Fid es único en un cluster de servidores Coda.

Coda tiene servidores de replicación de lectura/escritura. La ventaja de esto es la alta disponibilidad de los datos: Si un servidor falla, otros toman su lugar sin que un cliente note la falla.

Los volúmenes pueden ser almacenados en un grupo de servidores llamado VSG (*Volume Storage Group*).
Para volúmenes replicados, el VolumeId es un VolumeId replicado. El ID del volumen replicado trae junto un *Volume Storage Group* y un volumen lógico en cada uno de los miembros.

- El VSG es una lista de servidores que tienen una copia del volumen replicado.
- El volumen local para cada servidor define una partición, y un ID de volumen local contiene los archivos y meta-datos sobre este servidor.

Cuando Venus desea acceder un objeto sobre los servidores, primero necesita encontrar el VolumeInfo del volumen que contiene el archivo. Esta información contiene la lista de servidores y los IDs de volumen local de cada servidor donde el volumen es conocido. Para archivos, la comunicación con los servidores en un VSG es "*read-one, write-many*", esto es, lee el archivo desde un solo servidor en un VSG y propaga actualizaciones a todos los miembros VSG disponibles; el AVSG. Coda puede emplear RPCs multi-cast, de aquí que las actualizaciones write-many no causan un problema de desempeño. Las actualizaciones no son propagadas a todos los servidores, solo a los miembros de AVSG.

Antes de traer un objeto y sus atributos, Venus requerirá los sellos de versión (*version stamps*) de todos los servidores disponibles. Si detecta que algunos servidores no tienen las últimas copias de los archivos, se inicia un proceso de resolución, el cual trata de resolver automáticamente las diferencias. Si esto falla, el usuario debe reparar la falla manualmente. La resolución, aunque es iniciada por un cliente, es manejada enteramente por los servidores.

3.3.6 Escalabilidad

Un sistema distribuido escalable es uno que puede fácilmente adaptarse a la adición de nuevos usuarios y sitios. El crecimiento tiene consecuencias económicas, de desempeño y administrativas. La meta de Coda fue construir un sistema cuyo crecimiento incurriría en poco costo, degradación de desempeño y complejidad administrativa [Satyanarayanan *et al.*, 1990].

Coda preservó muchas de las características de su antecesor AFS. Para maximizar la función cliente-servidor, la mayoría de la carga la llevan los clientes. Solo las funciones esenciales de integridad o seguridad son ejecutadas por los servidores. Caching es la clave de la escalabilidad en AFS.

La naturaleza altamente dinámica de AFS mejora su escalabilidad. Una estación de trabajo con un disco pequeño puede acceder cualquier archivo en AFS por nombre. El usuario puede moverse a cualquier estación de trabajo y sin esfuerzo acceder cualquier archivo del sistema desde su nueva ubicación.

Las estaciones pueden ser apagadas o físicamente reubicadas en cualquier momento sin causar inconvenientes a otros usuarios. Para monitorear y dar servicio a los servidores AFS, solamente se requiere un pequeño staff operacional. El respaldo se realiza únicamente en los servidores, debido a que los discos de las estaciones de trabajo son solamente usados como caché. Los archivos pueden ser fácilmente movidos entre servidores durante la operación normal, sin afectar a los usuarios.

Coda retiene muchas de las características de AFS que contribuyen a la escalabilidad y seguridad:

- Los clientes colocan en caché de disco archivos enteros. Una vez que un archivo ingresa a la caché y abierto en un cliente, es inmune a las fallas en el servidor y en la red.

- La coherencia de caché es mantenidas por el uso de *callbacks*.

- Los clientes mapean archivos a servidores directamente y guardan en caché esta información.

- Usa autenticación basada en tokens y encriptación punto-a-punto integrada con su mecanismo de comunicación.

3.3.7 Alta disponibilidad en Coda

Coda se ha esforzado por implementar un mecanismo que permita la disponibilidad de datos en caso de alguna falla en cualquier parte del sistema [Satyanarayanan, 1990b].

Coda proporciona tolerancia a fallos por medio de dos mecanismos diferentes: La primera es la replicación de servidor, lo cual conste en almacenar copias de los archivos de un servidor en múltiples servidores, para crear un repositorio de almacenamiento compartido altamente disponible. La segunda es la operación desconectada, que entra en operación cuando ningún servidor puede ser contactado, de tal manera que el cliente maneja solamente los datos en caché.

Durante la fase de pruebas de la presente tesis, se usó la replicación de servidores en algunos casos de estudio, razón por la cual nos enfocaremos en la descripción de este mecanismo para lograr alta disponibilidad.

3.3.7.1 Replicación de servidores

La unidad de replicación en Coda es el volumen, un conjunto de archivos y directorios ubicados en un servidor y formando un subárbol parcial del espacio de nombres compartido. Cada archivo y directorio en Coda tiene un identificador de archivos único de bajo nivel (FID), un componente de cual identifica el volumen padre. Todas las replicas de un objeto tienen el mismo FID.

El conjunto de servidores con replicas de un volumen constituye un *Volume Storage Group* (VSG). El grado de replicación y la identidad de los sitios de replicación son especificados cuando se crea un volumen y son almacenados en una base de datos de replicación de volúmenes que están presentes en cada servidor. Aunque estos parámetros pueden ser cambiados posteriormente, no se espera que se hagan cambios frecuentes. Venus (el manejador de caché del cliente) mantiene la información del subconjunto del VSG que es accesible. Este subconjunto es llamado el grupo de volúmenes de almacenamiento accesibles (*Accesible Volume Storage Group*, AVSG) [Satyanarayanan M. *et al.*, 1990].

3.3.7.1.1 Estrategia

La estrategia de replicación que usa Coda es una variante del método *read-one write-all*. Cuando se esta sirviendo una falta de datos en el caché, el cliente obtiene datos de un miembro de su AVSG llamado el *servidor preferido*. El servidor preferido puede ser elegido de manera aleatoria o en base a un criterio de rendimiento, tal como la proximidad física, la carga del servidor o el poder de procesamiento del CPU del servidor. Aunque los datos son transferidos solo desde un servidor, los otros servidores son contactados por el cliente para verificar que el servidor preferido tiene la última copia de los datos. Si este no es el caso, el miembro del AVSG con la última copia toma el papel de sitio preferido; los datos son re-enviados y el AVSG es notificado que algunos de sus miembros tienen replicas antiguas. Como efecto secundario, se establece un *callback* con el servidor preferido.

Cuando un archivo es cerrado después de una modificación, esta es transferida en paralelo a todos los miembros de AVSG. La carga del CPU del servidor se minimiza, porque la carga de la propagación de datos esta sobre el cliente, en vez de en el servidor. Esto mejora la escalabilidad,

debido a que el CPU del servidor es el cuello de botella en muchos sistemas de archivos distribuidos.

3.3.7.1.2 Coherencia de caché

Las garantías que actualmente ofrece Coda requieren que un cliente reconozca tres clases de eventos no después de τ segundos después de que estos ocurran:

- La ampliación de un AVSG (implica accesibilidad de un servidor previamente inaccesible)
- La reducción de una AVSG (implica la inaccesibilidad hacia un servidor previamente accesible)
- Un evento de pérdida de *callback*

Venus detecta la ampliación de un AVSG, tratando de contactar miembros inaccesibles de un VSG, una vez cada τ segundos. Si un AVSG se amplia, los objetos en caché del volumen podrían ya no ser la última copia en el nueva AVSG, por lo que el cliente borra todas las *callbacks* relacionadas con estos objetos. La siguiente referencia a cualquiera de los objetos nuevos causará que se contacte el AVSG ampliado y se envié una nueva copia del AVSG.

Venus detecta reducción de un AVSG verificando sus miembros cada τ segundos. La reducción es detectada antes si una operación normal sobre AVGS falla. Si la reducción es causada por la perdida del servidor preferido, Venus desecha las *callbacks* que le pertenecían. Ya que de otra forma, permanecerían inválidas. Es importante notar que Venus solamente prueba los servidores de los cuales ha almacenado datos en caché; no prueba otros servidores ni otros clientes.

Si Venus hiciera *callbacks* a todos los miembros de su AVSG, la prueba para detectar la reducción de AVSG también detectaría eventos de perdida de *callbacks*. Debido a que mantener el estado de la *callback* en todos los servidores es costoso, Venus solamente mantiene una *callback* en el servidor preferido. La prueba al servidor preferido detecta los eventos de pérdida de *callbacks*.

Sin embargo, mantener callbacks solo en un servidor introduce un nuevo problema. El servidor preferido para un cliente, no necesariamente existe en el AVSG de otro cliente. Así, una actualización a un objeto por el segundo cliente podría no causar una *callback* para el objeto en el primer cliente que no sería realizada.

Para detectar actualizaciones no realizadas por el servidor preferido, cada prueba de Venus requiere la versión del vector del volumen (*volume CVV*), para cada volumen desde el cual se trae información a caché. Un *volume CVV* contiene información de actualización acerca del volumen entero y es actualizado como efecto secundario en cada operación que modifica el volumen. Un desajuste en los *volume CVV's* indican que algunos miembros AVSG omitieron una actualización. Aunque las actualizaciones omitidas podrían no haber sido a un objeto en el caché, Venus desecha sus callbacks sobre todos los objetos del volumen.

3.3.8 Seguridad en Coda

Coda cumple con las bases de seguridad de su antecesor AFS. Andrew proporciona mecanismos de seguridad, los cuales, por si solos no garantizan la seguridad del sistema; por lo que se deben seguir procedimientos administrativos y operacionales [Satyanarayanan, 1990b].

Dominio de protección. El dominio de protección esta compuesto de usuarios y grupos. Un usuario es una entidad, usualmente humana, que puede autentificarse a si misma con Vice, y ser responsable de sus acciones. Un *grupo* es un conjunto de otros grupos y usuarios. Cada grupo es asociado con un usuario único llamado su propietario (*owner*).

Existe un grupo de nombre System:Administrators. La membresía en este grupo asigna privilegios administrativos, incluyendo acceso irrestricto a cualquier archivo en el sistema.

Autentificación. El mecanismo de RPC de Andrew proporciona soporte para comunicación segura y autentificada entre clientes y servidores, usando una variante del algoritmo de llave privada de Needham y Schroeder. Cuando un usuario ingresa en una estación de trabajo, su contraseña es usada para obtener tokens desde un servidor de autentificación. Estos tokens son servidos por Venus y usados como necesarios para establecer conexiones RPC seguras hacia servidores de archivos, identificándose como un usuario que pertenece al sistema.

Las contraseñas no son guardadas en las estaciones de trabajo, debido a que los tokens expiran después de 24 horas.

Protección del sistema de archivos. Andrew usa un mecanismo de lista de acceso para protección de archivos. Los derechos totales especificados para un usuario son la unión de los derechos especificados para el usuario y para los grupos al que pertenece. Las listas de acceso están asociadas con directorios en vez de con archivos individuales.

3.4 Conclusiones

En este Capítulo se han presentado las herramientas computacionales que son utilizadas en el desarrollo de esta Tesis. Se han señalado que MySQL Cluster además de ser un sistema manejador de bases de datos distribuidas de código abierto posee las características que brindan al usuario final una de alta disponibilidad e integridad de la información, tales como tolerancia a fallos, recuperación de nodos y replicación de datos. MySQL Cluster ha logrado confiabilidad en los datos apoyándose en el protocolo de compromiso de dos fases asegurando la propiedad de atomicidad en las transacciones y en los protocolos de checkpoint global y checkpoint local para asegurar las propiedades de consistencia, aislamiento y durabilidad que deben cumplir las bases de datos distribuidas para conservar la integridad de la información. También se describió que MySQL Cluster logra su alta disponibilidad en base a su esquema de replicación de información de nodos de datos.

Se describieron las características del sistema de archivos distribuidos Coda. Coda también es una herramienta de código abierto, cuyo antecesor fue Andrew File System [Satyanarayanan, 1990b] Se indicó que Coda proporciona el mecanismo de replicación se servidores para crear alta disponibilidad de sus datos, además de que proporciona seguridad de acceso a los datos.

Tanto MySQL Cluster como Coda, proporcionan un ambiente de alta disponibilidad, implementación sencilla y no requieren de hardware especial para crear un ambiente distribuido, razones por las cuales se seleccionaron para realizar los Casos de Estudio de esta Tesis.

Capítulo 4
Manejo de Sistemas Distribuidos de Cómputo

En el presente Capítulo se describe la configuración del sistema distribuido implementado, el equipo usado durante los Casos de Estudio y en qué consistió cada uno de los Casos de Estudio.

4.1. Introducción

La tecnología de bases de datos distribuidas representa uno de los desarrollos más relevantes de las últimas dos décadas. Durante este periodo, los temas de investigación acerca de bases de datos distribuidas han sido objeto de un intenso estudio. Se espera que la tecnología de bases de datos distribuidas tenga impacto en el procesamiento de datos de la misma forma que los sistemas centralizados lo hicieron hace dos décadas [Özsu y Valduriez, 1991].

Debido a la demanda de disponibilidad y autonomía en los sistemas computacionales y a los avances que se han logrado en tecnología de bases de datos y comunicaciones; se ha extendido la aplicación y uso de los sistemas de bases de datos distribuidas. Actualmente, muchos sistemas manejadores de bases de datos soportan extensiones para hacer proceso distribuido. Los diseñadores de aplicaciones de bases de datos distribuidas enfrentan un nuevo y relevante problema: Como distribuir los datos y programas en diferentes computadoras para obtener un desempeño, confiabilidad y disponibilidad eficientes. El diseño de la distribución está emergiendo como un nuevo problema en el área de diseño de bases de datos, el cual requiere su propia teoría, metodologías de diseño y utilerías de soporte [Ceri *et al.*, 1987].

El grado de sofisticación de los sistemas manejadores de bases de datos distribuidas (SMBDD) es frecuentemente medido por el grado de transparencia de ubicación que proporciona a los usuarios [Özsu y Valduriez, 1999]. En una situación ideal, los usuarios no necesitan estar conscientes de la distribución de los datos, y el sistema asume la responsabilidad de distribuir las operaciones de acceso hacia los diferentes sitios de bases de datos. Sin embargo, la distribución actual de los datos afecta el rendimiento general del sistema: El tiempo y costo requerido para acceder múltiples objetos de datos difiere grandemente dependiendo de si todos los objetos están almacenados en el mismo sitio o están dispersos en múltiples sitios. La replicación de datos afecta la confiabilidad y disponibilidad del sistema, porque varias copias de la misma información se convierten en disponibles con modos de falla independientes. Al mismo tiempo, la replicación de datos afecta el rendimiento, debido a los requerimientos de mantenimiento de consistencia de las copias. Así, en la actualidad, el diseñador de la base de datos debe considerar la distribución de los datos, aún cuando el sistema soporte un alto grado de transparencia [Ceri *et al.*, 1987].

Un principio clave en el diseño de sistemas distribuidos es lograr lo máximo posible de datos y aplicaciones ubicados de forma local. Mientras las bases de datos distribuidas habilitan una comunicación más sofisticada entre sitios, la mayor motivación para desarrollar una base de datos distribuida es reducir la comunicación, ubicando los datos tan cerca como sea posible a las aplicaciones que los usan. De tal manera que, en una base de datos bien diseñada “el 90 por ciento de los datos deberían ser encontrados en el sitio local y solamente el 10 por ciento de los datos debería ser accedidos del sitio remoto” [Ceri *et al.*, 1987]. Un diseño que resulta efectivo es uno que asegure el lugar para el más grande número de aplicaciones. Por otro lado, un número creciente de pequeñas y medianas organizaciones están ajustando sus bases de datos a los sistemas de código abierto, los cuales tienden a costar menos que sus contrapartes vendidas por IBM, Microsoft y Oracle. Aunque el software de código abierto enfrenta en muchas ocasiones el problema de falta de soporte técnico, en la actualidad muchas compañías se están preocupando por solucionar este problema creando documentación que se encuentra disponible en línea o foros de consulta, tal es el caso de MySQL AB.

Las mayores bases de datos de código abierto incluyen Firebird, MaxDB, MySQL, PostgreSQL y Sleepycat Software de Berkeley DB [Dailey, 2004].

Compañías tales como Google y Travelocity usan estos productos como bases de datos en sus aplicaciones. Nokia usa Berkeley DB para almacenamiento y recuperación para el servicio de mensajes cortos [Forum Nokia, 2007]. Westone Laboratorios, fabricante de aparatos para oído, usa MySQL como su base de datos con la información del paciente [MySQL Press Releases, 2003].

Aunque la popularidad esta creciendo, las bases de datos abiertas enfrentan varios retos para triunfar en el mercado. Por ejemplo, se tienen algunos problemas escalando en términos de volumen de datos y número de usuarios [Dailey, 2004].

4.2. Descripción general del sistema propuesto

A continuación se presenta la implementación de una base de datos distribuida utilizando dos plataformas diferentes. Para la construcción de ambas plataformas se usó MySQL Cluster 5.1 [MYSQL, 2008] instalado sobre un sistema operativo Ubuntu 7.04 [UBUNTU, 2008].

- En la primera plataforma, la implementación se realizó utilizando MySQL Cluster, el cual integra el motor de almacenamiento NDB junto con MySQL estándar.
- En la segunda plataforma, la implementación de la base de datos distribuida se realizó combinando dos tecnologías como la facilidad que proporciona MySQL 5.1 para soportar tablas particionadas [Schumacher, 2006] y el sistema de archivos distribuidos Coda 6.9.3 [Satyanarayanan *et al.*, 1990]. El esquema de configuración se muestra en la Figura 4.2.

Tanto Ubuntu 7.04, como MySQL Cluster 5.1 como Coda 6.9.3 que presentan las ventajas de software de código libre como lo son el bajo costo de implementación y la posibilidad de modificar el código fuente si se requiere.

En los casos de estudio presentados se hace un análisis del desempeño de cada plataforma y un análisis comparativo entre ambas.

En ambas plataformas se usó la base de datos de ejemplo *world* [MYSQL, 2008], que se encuentra a disposición de los usuarios de MySQL en la página web de descarga de software de este sistema manejador de bases de datos. Esta base de datos contiene tres tablas idénticas a las mostradas en la Tabla 4.1. Las características de dichas tablas se presentan en la Tabla 4.2.

Tabla 4.1 Características de la base de datos world.

Nombre de la Tabla	Número de columnas	Amplitud en bytes	Llave primaria	Llave secundaria
City	5	178	Si	Sí
Country	15	490	Si	No
CountryLanguage	4	51	Si	No

Tabla 4.2 Características de las tablas de la base de datos world

Tabla City

Nombre del campo	Tipo MySQL	Tamaño
ID	int	11
Name	char	35
CountryCode	char	3
District	char	20
Population	int	11

Tabla CountryLanguage

Nombre del campo	Tipo MySQL	Tamaño
CountryCode	char	3
Language	char	30
IsOfficial	enum	2
Percentage	float	4,1

Tabla Country

Nombre del campo	Tipo MySQL	Tamaño
Code	char	3
Name	char	52
Continent	enum	7
Region	char	26
SurfaceArea	float	10,2
IndepYear	smallint	6
Population	int	11
LifeExpectancy	float	3,1
GNP	float	10,2
GNPOld	float	10,2
LocalName	char	45
GovernmentForm	char	45
HeadOfState	char	60
Capital	int	11
Code2	char	2

Se ejecutaron cuatro pruebas diferentes para cada caso de estudio.

Para cada una de las pruebas realizadas se requirió el uso de la base de datos world. La tabla City, fue variando en número de registros para cada prueba; el contenido de tales registros fue generado de manera aleatoria.

Para cada prueba se realizó el registro de 30 resultados diferentes. La prueba inicial de cada caso de estudio se realizó con 66,666 registros y en cada se prueba se incrementó el número de registros en 66,666, de tal forma que en todos los casos la prueba número 30 se realizó con 1,999,980 registros.

Las pruebas realizadas en ambas plataformas consistieron en:

- La medición del tiempo necesario para ejecutar un conjunto las inserciones a la base de datos distribuida.

- La medición de los tiempos para ejecutar una consulta secuencial a una tabla de la base de datos. La consulta 1 fue: SELECT * FROM City;

- La medición de los tiempos para realizar una consulta que involucra dos bases de datos. La consulta 2 fue: SELECT City.ID, City.Name, City.Population, Country.Name, Country.Continent FROM City JOIN Country ON (City.CountryCode=Country.Code);

- La medición de los tiempos para realizar una consulta que involucra todas las tablas de la base de datos, siendo la consulta 3: SELECT City.ID, City.Name, Country.Name, Country.Continent, CountryLanguage.Language FROM City JOIN (Country CROSS JOIN CountryLanguage) ON (City.CountryCode=Country.Code AND Country.Code=CountryLanguage.CountryCode) WHERE Country.Continent='Asia';

La medición del tiempo para el caso de las inserciones a la base de datos se realizó mediante la instrucción *time* del sistema operativo, en el caso de las consultas el tiempo registrado es calculado por MySQL al ejecuta una consulta (no toma en consideración el tiempo de despliegue de resultados).

Se realizaron seis Casos de Estudio, divididos en dos partes; los tres primeros casos de estudio consideran 4 nodos de datos para constituir la base de datos distribuida. Este cluster de computadoras consiste de una computadora portátil Dell Latitude D410 con procesador Pentium M (Centrino) y cuatro computadoras personales Compaq Presario con procesador Celeron D3.46 GHz.

En la segunda etapa los casos de estudio se realizaron con 10 nodos de datos, configurados en el equipo de cómputo que se presenta en la Figura 4.2 y que está constituido por un servidor HP Proliant DL360 G5, con 2 procesadores Quad-Core Xeon 2.0 GHz, un servidor HP Proliant ML350 G3 con dos procesadores Dual-Core Xeon 3.06 GHz, y tres computadora personales Compaq Presario con procesador Celeron D3.46 GHz.

Esta configuración ha sido posible ya que las características de hardware de los equipos servidores permiten la configuración de más de un nodo de datos en una misma computadora, conservando un ambiente que proporciona disponibilidad y confiabilidad en la información.

4.2.1 Configuración MySQL Cluster

a) Configuración con 4 nodos de datos

La configuración realizada para el cluster MySQL se muestra en la Figura 4.1 y consta de un nodo Administrador (MGM), un nodo MySQL y cuatro nodos de datos dentro de una red local con capacidad de transmisión de 1 Gigabit por segundo.

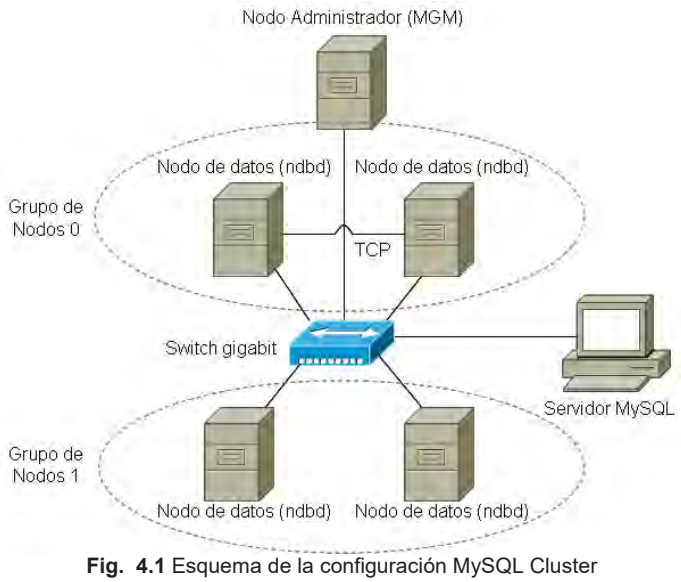


Fig. 4.1 Esquema de la configuración MySQL Cluster

Además de los parámetros necesarios para el funcionamiento adecuado del cluster MySQL, se incluyó en el archivo *config.ini* dentro del manejador del cluster, los parámetros para incluir una conexión TCP directa entre dos nodos de datos. En la configuración realizada, se tomó en cuenta la creación de 2 réplicas por partición, lo cual generó dos grupos de nodos dentro del cluster.

El cluster MySQL se formó con cinco computadoras personales con las características que se indican en la Tabla 4.3.

Tabla 4.3 Características del equipo de cómputo de la plataforma MySQL Cluster

Tipo de Nodo	Nombre	Procesador	RAM	Disco Duro
(MGM)	galois	Celeron D 3.46 GHz	1.5 GB	160 GB
MySQL	vmedina-laptop	Pentium M (Centrino) 2.00 GHz	2 GB	80 GB
Datos	vmedina-laptop	Pentium M (Centrino) 2.00 GHz	2 GB	80 GB
Datos	fermat	Celeron D 3.46 GHz	1.5 GB	160 GB
Datos	descartes	Celeron D 3.46 GHz	1.5 GB	160 GB
Datos	lhopital	Celeron D 3.46 GHz	1.5 GB	160 GB

En la Figura 4.2 se muestra el equipo que constituye el sistema de cómputo distribuido durante la primera parte de los casos de Estudio.

Fig. 4.2 Equipo de pruebas para la primera fase

a) Configuración con 10 nodos de datos

La configuración realizada para el cluster MySQL se muestra en la Figura 4.3 y consta de un nodo Administrador (MGM), dos nodos MySQL y diez nodos de datos dentro de una red local con capacidad de transmisión de 1 Gigabit por segundo.

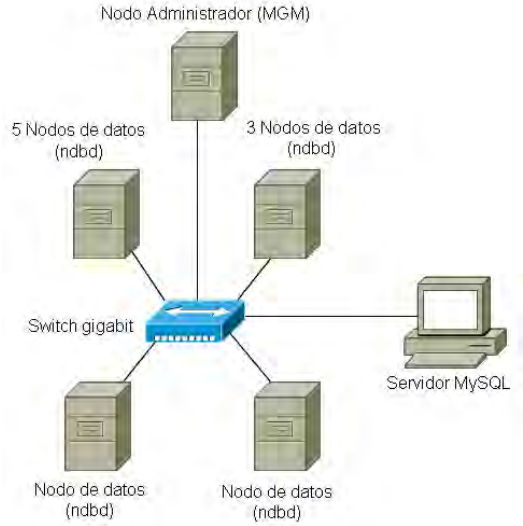


Fig. 4.3 Esquema de la configuración MySQL Cluster con 4 nodos de datos

En el archivo *config.ini* dentro del manejador del cluster se incluyeron los parámetros necesarios para la creación de 10 nodos de datos y 2 réplicas por cada partición de las tablas con el motor NDB. El cluster MySQL contiene los nodos de datos que se indican en la Tabla 4.4

Tabla 4.4 Nodos de datos por equipo

Tipo de Nodo	Nombre	Nodos de datos
(MGM)	galois	0
Datos / MySQL	rvaro	5
Datos / MySQL	biblioteca	3
Datos	fermat	1
Datos	descartes	1

El cluster MySQL se formó con 2 servidores y 3 computadoras personales con las características que se indican en la Tabla 4.5.

Tabla 4.5 Características del equipo de cómputo de la plataforma MySQL Cluster

Tipo de Nodo	Nombre	Procesador	RAM	Disco Duro
(MGM)	galois	Celaron D 3.46 GHz	1.5 GB	160 GB
Datos / MySQL	rvaro	2 Quad-Core Intel Xeon 2.00 GHz	3 GB	72 GB
Datos / MySQL	biblioteca	2 Dual-Core Intel Xeon 3.06 GHz	2.5 GB	20 GB
Datos	fermat	Celeron D 3.46 GHz	1.5 GB	160 GB
Datos	descartes	Celeron D 3.46 GHz	1.5 GB	160 GB

En la Figura 4.4 se presenta el equipo de cómputo que se usó durante la segunda etapa de los Casos de Estudio.

Fig. 4.4 Equipo de pruebas de la segunda etapa.

4.2.2 Configuración Coda

a) Configuración con 4 nodos de datos

La configuración del sistema de archivos distribuidos Coda, consta de un servidor SCM, cuatro servidores no SCM y un cliente Coda, según se ilustra en la Figura 4.5.

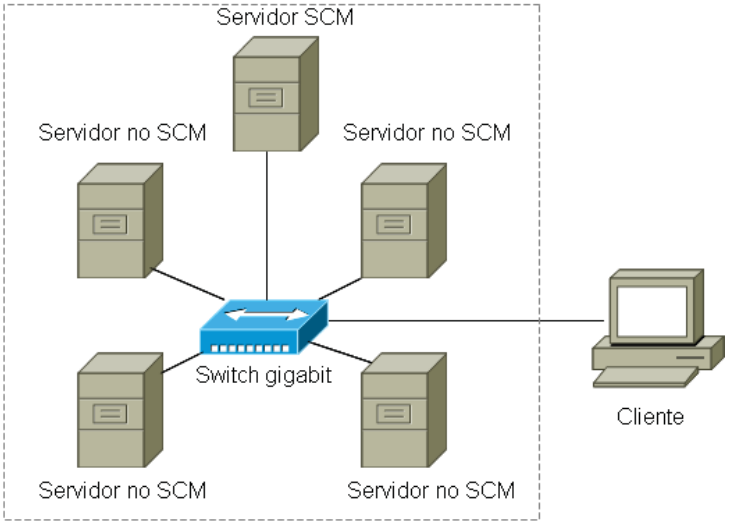


Fig. 4.5 Configuración Coda en con 4 nodos de datos

Las características del equipo de cómputo empleado para instalar la plataforma MySQL/Coda se muestran en la Tabla 4.6.

Tabla 4.6 Características del equipo de cómputo de la plataforma MySQL/Coda

Tipo de Nodo	nombre	procesador	RAM	Disco Duro
Servidor SCM	vmedina-laptop	Pentium M (Centrino)	2 GB	80 GB
Cliente Coda	galois	Celeron D 3.46 GHz	1.5 GB	160 GB
Servidor no SCM	fermat	Celeron D 3.46 GHz	1.5 GB	160 GB
Servidor no SCM	descartes	Celeron D 3.46 GHz	1.5 GB	160 GB
Servidor no SCM	laplace	Celeron D 3.46 GHz	1.5 GB	160 GB
Servidor no SCM	lhopital	Celeron D 3.46 GHz	1.5 GB	160 GB

Cada servidor ordinario tiene replicado el volumen de datos donde se almacena una partición MySQL. La Figura 4.6 muestra la configuración de los volúmenes de datos Coda para los distintos servidores existentes; cada volumen es replicado a otro servidor no SCM, para crear un esquema de redundancia que permita alta disponibilidad. Las particiones de la base de datos fueron creadas por medio de la función *hash* que existe por omisión; usando particionamiento horizontal, el motor de almacenamiento de la base de datos es MyISAM.

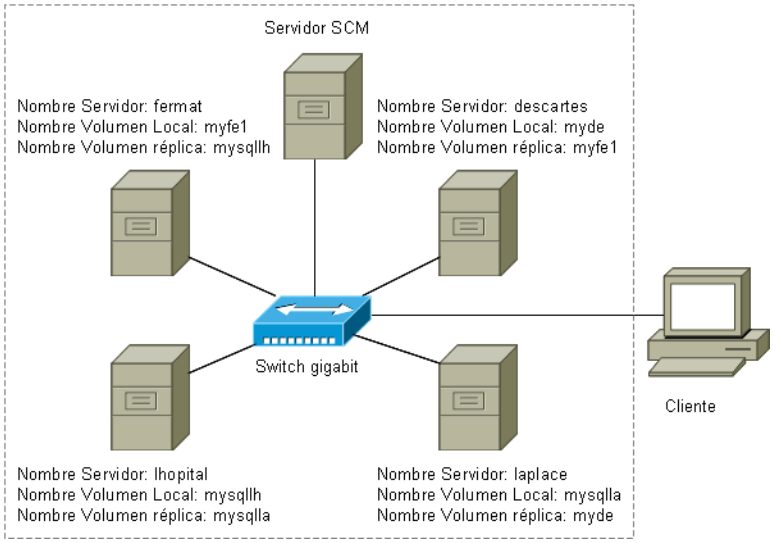


Fig. 4.6 Esquema de volúmenes replicados en Coda con 4 nodos de datos

a) Configuración con 10 nodos de datos

La configuración del sistema de archivos distribuidos Coda, consta de un servidor SCM, cuatro servidores no SCM y un cliente Coda, según se ilustra en la Figura 4.7.

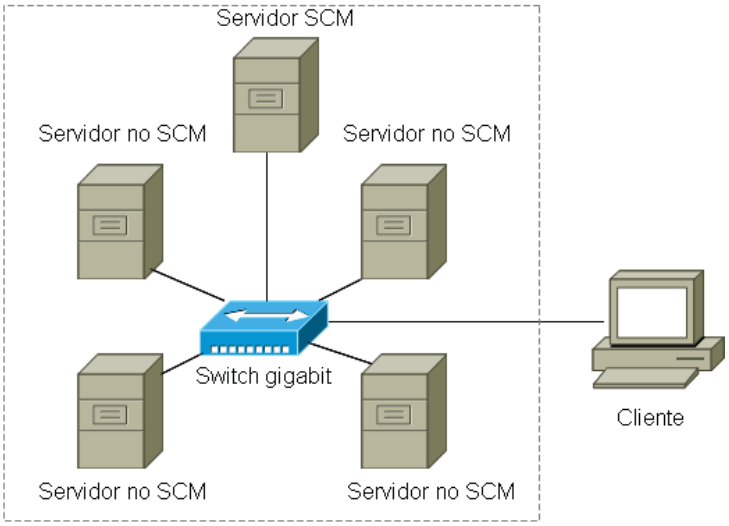


Fig. 4.7 Configuración Coda con 10 volúmenes de datos

Las características del equipo de cómputo empleado para instalar la plataforma MySQL/Coda se muestran en la Tabla 4.7.

Tabla 4.7 Características del equipo de cómputo de la plataforma MySQL/Coda

Tipo de Nodo	nombre	procesador	RAM	Disco Duro
Servidor SCM	vmedina-laptop	Pentium M (Centrino)	2 GB	80 GB
Cliente Coda	galois	Celeron D 3.46 GHz	1.5 GB	160 GB
Servidor no SCM	rvaro	2 Quad-Core Intel Xeon 2.00 GHz	3 GB	72 GB
Servidor no SCM	biblioteca	2 Dual-Core Intel Xeon 3.06 GHz	2.5 GB	20 GB
Servidor no SCM	fermat	Celeron D 3.46 GHz	1.5 GB	160 GB
Servidor no SCM	descartes	Celeron D 3.46 GHz	1.5 GB	160 GB

Cada servidor ordinario tiene replicado el volumen de datos donde se almacena una partición MySQL. La Tabla 4.8 muestra la configuración de los volúmenes de datos Coda para los distintos servidores existentes; cada volumen es replicado a otro servidor no SCM, para crear un esquema de redundancia que permita alta disponibilidad. Para este caso, las particiones de la base de datos también fueron creadas por medio de la función *hash* que existe por omisión, usando particionamiento horizontal, el motor de almacenamiento de la base de datos es MyISAM.

Tabla 4.8 Configuración de volúmenes replicados en Coda con 10 nodos de datos

Nombre del volumen	Computadora origen	Se replica en
rva1	rvaro	biblioteca
rva2	rvaro	biblioteca
rva3	rvaro	biblioteca
rva3	rvaro	fermat
rva5	rvaro	descartes
bib1	biblioteca	fermat
bib2	biblioteca	descartes
bib3	biblioteca	rvaro
fer1	fermat	descartes
des1	descartes	rvaro

4.3 Cálculo de la eficiencia

En cada uno de los casos de estudio considerados en esta Tesis, se realizó la evaluación de la eficiencia de una plataforma respecto de la otra de dos maneras. La primera, consistió en hacer un análisis de los datos obtenidos directamente durante las pruebas para las dos plataformas propuestas, tanto para MySQL Cluster como para MySQL/Coda, donde se encontró el mayor punto de eficiencia de las 30 ejecuciones realizadas en cada caso.

La eficiencia, medida en términos de la fracción de tiempo que los procesadores consumen haciendo un trabajo útil, es una relación métrica que puede algunas veces proveer de una cantidad más conveniente para medir el desempeño de un algoritmo paralelo. Esta medida caracteriza la efectividad con la que un algoritmo utiliza los recursos computacionales de alguna computadora paralela de manera que sea independiente del tamaño del problema. La eficiencia relativa se define como [Ramos-Paz, 2007]:

$$Ef = \frac{t_a}{t_b} \qquad (4.1)$$

Ef = Eficiencia relativa

t_a = Tiempo de a en segundos

t_b = Tiempo de b en segundos

Donde si $t_a > t_b$, entonces b es Ef veces más rápido que a; y si $t_b > t_a$, entonces a es $\frac{1}{Ef}$ veces más rápido que b.

También se calculó la eficiencia de forma analítica, haciendo uso de las ecuaciones que modelan el comportamiento de los tiempos de respuesta en cada caso. Si se considera que en cada caso presentado las rectas tienen su origen en 0, entonces las rectas tendrían la forma (4.2)

$$y = mx \qquad (4.2)$$

En el comparativo de eficiencia realizado en cada caso de estudio, se tomó en consideración el ajuste de las dos curvas producidas por los tiempos de respuesta (segundos) de cada prueba, de donde cada recta se expresa en función de una constante y del número de prueba ejecutada, como se muestra en la Figura 4.8.

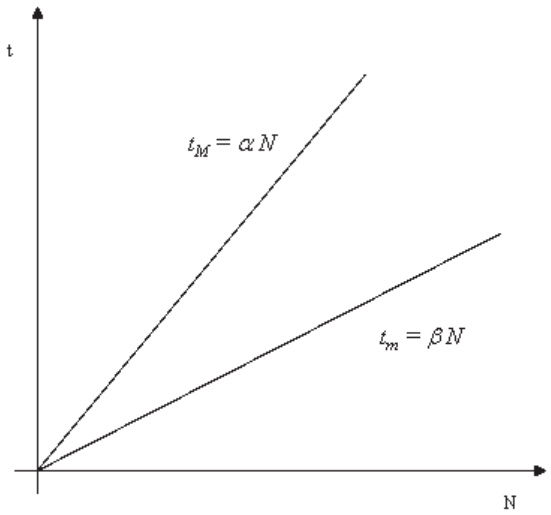


Fig. 4.8 Dos rectas en el origen

donde,

t_M = Curva con tiempos mayores
α = Pendiente de la recta que representa los tiempos mayores
t_m = Curva con tiempos menores
β = Pendiente de la recta que representa los tiempos menores
N = Número de prueba ejecutada

de tal forma que, la eficiencia relativa *s*, puede calcularse con la ecuación (4.3)

$$s = \frac{T_M}{T_m} = \frac{\alpha N}{\beta N} = \frac{\alpha}{\beta} \qquad (4.3)$$

donde, si T_M pertenece al proceso que consume los tiempos mayores de respuesta, entonce, *s* será el número de veces que T_m es más eficiente.

4.4 Conclusiones

En el presente Capítulo se ha descrito como se configuró el equipo de cómputo sobre el que se operó la base de datos distribuida. Se listaron las características físicas de los equipos que se utilizaron en las dos partes que conforman los Casos de Estudio; así como también la distribución de los nodos de datos en cada diferente configuración. Se señaló que aplicaciones de código abierto intervinieron en las dos plataformas de bases de datos distribuidas y la manera de evaluar la eficiencia relativa entre las dos diferentes plataformas, donde la variable considerada es el tiempo de respuesta en segundos.

Capítulo 5
Casos de Estudio

5.1 Introducción

Una vez que se ha instalado y configurado el hardware y software necesarios para la operación del cluster sobre el cual se alojará la base de datos distribuida en las dos plataformas, MySQL Cluster y la combinación de MySQL estándar con el sistema de archivos distribuido Coda; se presentan seis casos de estudio en dónde se muestra la aplicación técnica de la propuesta de esta Tesis. En la primera parte, se analiza la operación de la base de datos distribuida a través de 4 nodos de datos y en la segunda parte, la base de datos se encuentra distribuida en 10 nodos de datos. Durante las pruebas en estos casos de estudio se realizan pruebas de eficiencia en tiempo de respuesta para 4 diferentes operaciones y la simulación de fallas en comunicación, dejando no disponibles a algunos nodos de la base de datos distribuida.

En cada uno de los casos de estudio realizados en esta Tesis se aplicaron cuatro tipos de pruebas:

- La medición del tiempo de respuesta en segundos requerido para insertar cada uno de los grupos de registros a la base de datos distribuida.

- Consulta 1: La medición de los tiempos de respuesta en segundos para ejecutar una consulta que requiere el barrido de toda la base de datos. La consulta es SELECT * FROM City.

- Consulta 2: La medición de los tiempos de respuesta en segundos para realizar una consulta que recupera la información de dos bases de datos: SELECT City.ID, City.Name, City.Population, Country.Name, Country.Continent FROM City JOIN Country ON (City.CountryCode=Country.Code.

- Consulta 3: La medición de los tiempos de respuesta en segundos para realizar una consulta que recupera información de tres tablas de la base de datos: SELECT City.ID, City.Name, Country.Name, Country.Continent, CountryLanguage.Language FROM City JOIN (Country CROSS JOIN CountryLanguage) ON (City.CountryCode=Country.Code AND Country.Code=CountryLanguage.CountryCode) WHERE Country.Continent='Asia'.

En los datos obtenidos en las muestras de cada una de las pruebas realizadas en los casos de estudio que se describen en las siguientes secciones, se puede observar mediante una inspección visual de los datos sugieren una cierta tendencia, sin embargo se presenta cierta variabilidad; por lo cual se consideró necesario aplicar un método de ajuste de curvas para predecir el comportamiento de los datos a través del tiempo en una forma más precisa.

En todos los casos de estudio se ha incluido la ecuación que describe el comportamiento matemático de cada una de las pruebas realizadas, dicha ecuación se obtuvo con un ajuste de curvas mediante la función *polyfit* de Matlab, dicha función realizó el ajuste a una recta de la forma:

$$y = mx + b \qquad (5.1)$$

donde,

m = pendiente de la recta
b = ordenada al origen

Los resultados obtenidos de las pruebas realizadas muestran curvas que inicia en el origen, sin embargo, las rectas obtenidas mediante la función de Matlab muestran un valor para la ordenada al origen, lo cual es efecto del error numérico de la diferencia entre el valor observado y el valor ajustado que genera en método de mínimos cuadrados.

NOTA: Como se mencionó en el Capítulo 4, pág. 59, el número de registros en cada muestra es igual al número de la muestra multiplicado por 66,666. Dado que esta transformación es lineal, no afecta la linealidad de los resultados obtenidos en los experimentos.

5.2 Parte uno. 4 nodos de datos

Durante la primera parte de los casos de estudio, en la configuración de MySQL Cluster se utilizó un nodo administrador (MGM), como nodo MySQL y 4 nodos de datos (ndbd) instalados cada uno en una computadora personal con las mismas características del equipo del nodo administrador, las características física de los equipos se muestran en la Tabla 2.2., pag. 17.

En el caso de MySQL/Coda se utilizaró un servidor SCM, un cliente Coda y cuatro nodos servidores no SCM cuyas características física se listan en la Tabla 2.5., pag 19.

La interconexión de la red se realiza utilizando un switch de 1 Gb. La simulación de las fallas se realizó deteniendo el demonio *ndbd* encargado de mantener activo el nodo de datos, para el caso de las fallas en MySQL y en MySQL/Coda, las fallas se simularon apagando las computadoras que contienen los volúmenes de datos en falla.

5.2.1 Caso de Estudio 1. Todos los nodos activos

En el primer caso de estudio se incluyeron los cuatro nodos activos dentro del cluster, lo cual representa que la replicación se hará completa, es decir, sobre el nodo y/o volumen redundante, según la plataforma correspondiente, por lo que para cada fragmento se realizarán dos replicas de manera automática. Se obtuvieron los siguientes resultados y gráficas comparativas de tiempo de respuesta entre MySQL Cluster y MySQL/Coda descritas a continuación.

En la Figura 5.1, se observa el comportamiento del tiempo de respuesta que presentan las transacciones de inserción en un ambiente MySQL Cluster y en un ambiente que combina Coda con MySQL estándar. El ambiente MySQL/Coda logra mejores tiempos de respuesta, obteniéndose una mayor eficiencia de los tiempos logrados por MySQL/Coda con respecto a los de MySQL Cluster; dicha eficiencia llega a alcanzar un porcentaje máximo de hasta un 407.27%, durante la prueba número 30.

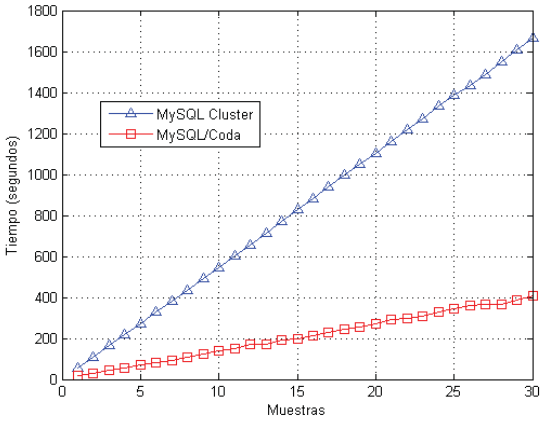


Fig. 5.1 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, inserción de registros, todos los nodos activos.

Según el ajuste de las curvas realizado con mínimos cuadrados se obtuvo que MySQL/Coda es más eficiente 4.115 veces que MySQL Cluster, este valor es cercano al obtenido con los datos reales. Las ecuaciones de la recta aproximada de cada plataforma se pueden apreciar en la Tabla 5.1.

Tabla 5.1 Ecuaciones de las rectas ajustadas para inserción de registros, con todos los nodos activos. Caso de estudio 1.

	Ecuación de la recta
MySQL Cluster	$y = 55.507x - 5.218$
MySQL/Coda	$y = 13.48x + 1.663$

En la gráfica mostrada en la Figura 5.2, se puede observar que el tiempo de respuesta del ambiente MySQL/Coda, comparado contra el tiempo de respuesta de MySQL Cluster, es menor, cuando se realiza una consulta que requiere un barrido completo de la tabla, como es el caso de la consulta 1, donde se realiza la recuperación de todo el contenido de una tabla.

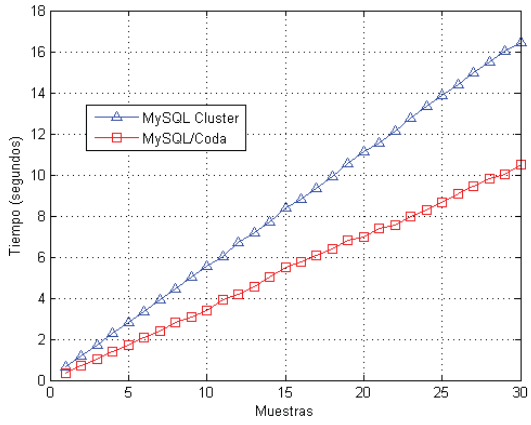


Fig. 5.2 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, consulta 1, todos los nodos activos

Durante esta prueba se encontró que MySQL/Coda puede ser hasta 1.56 veces más rápido en este tipo de consulta que MySQL Cluster, para el caso de 1,999,980 registros.

Después de realizar el ajuste de las curvas con los datos obtenidos en la consulta 1 para las dos plataformas analizadas, se obtuvo de la relación de eficiencia que MySQL/Coda presenta mayor eficiencia que MySQL Cluster en hasta 1.58 veces, lo cual, es un valor muy cercano a lo calculado con los datos reales. Las ecuaciones de las líneas ajustadas se presentan en la Tabla 5.2.

Tabla 5.2 Ecuaciones de las rectas ajustadas para la consulta 1, con todos los nodos activos. Caso de estudio 1.

	Ecuación de la recta
MySQL Cluster	$y = 0.55x + 0.0523$
MySQL/Coda	$y = 0.349x + 0.016$

En la Figura 5.3, se puede observar que el comportamiento de MySQL Cluster es considerablemente más eficiente, en cuanto a tiempo de respuesta obtenido para la consulta 2, que consiste en recuperar la información después de aplicar la operación JOIN a dos tablas de la base de datos, comparado con el tiempo de respuesta que arroja MySQL/Coda. Se registró que MySQL Cluster es hasta 1.33 veces más eficiente en tiempo de respuesta con respecto al tiempo obtenido MySQL/Coda. En este caso se puede apreciar que los métodos de recuperación de datos empleados por MySQL Cluster son más eficientes que los de MySQL estándar. De acuerdo con el optimizador de consultas para MySQL Cluster; se usó el método de acceso de índices ordenados, y MySQL estándar empleó una combinación de acceso de llave primaria y llave única.

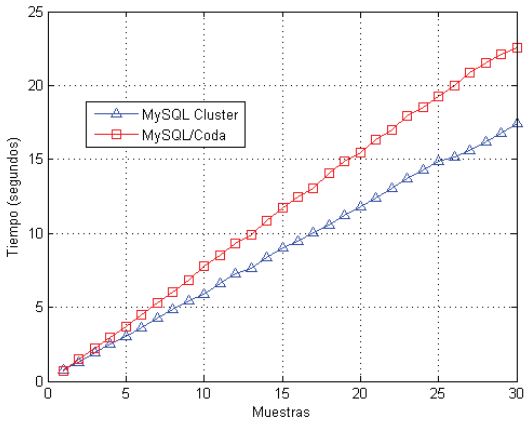


Fig. 5.3 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, consulta 2, todos los nodos activos

Después de realizar el ajuste de las curvas con mínimos cuadrados, se obtuvo de la relación de eficiencia entre las pendiente de ambas que MySQL Cluster es más eficiente que MySQL/Coda en hasta 1.339 veces en relación al tiempo de respuesta obtenido con ambas aplicaciones. En esta prueba, los datos reales y los ajustados por mínimos cuadrados tienen valores muy cercanos. Las ecuaciones de las rectas ajustadas se presentan en la Tabla 5.3.

Tabla 5.3 Ecuaciones de las rectas ajustadas para la consulta 2, con todos los nodos activos. Caso de estudio 1.

	Ecuación de la recta
MySQL Cluster	$y = 0.578x + 0.196$
MySQL/Coda	$y = 0.775x - 0.578$

En la Figura 5.4, se observan los tiempos de respuesta para realizar la consulta 3, donde la recuperación de la información se realiza desde las tres tablas relacionadas de la base de datos, mediante ambas plataformas. Se observa también que con MySQL/Coda, mientras mayor el es número de registros que contiene la base de datos, mayor es el tiempo de respuesta para la consulta 3. Por otro lado, el tiempo de respuesta de MySQL Cluster para todas las pruebas se mantiene casi constante, a pesar de que la consulta pareciera ser más complicada, por tener que conjuntar información de las tres tablas y que el número de registros involucrados se incrementa. De tal forma, que para este tipo de consulta, el tiempo de respuesta de MySQL Cluster llega a ser hasta 93.37 veces más rápido que el de la plataforma MySQL/Coda.

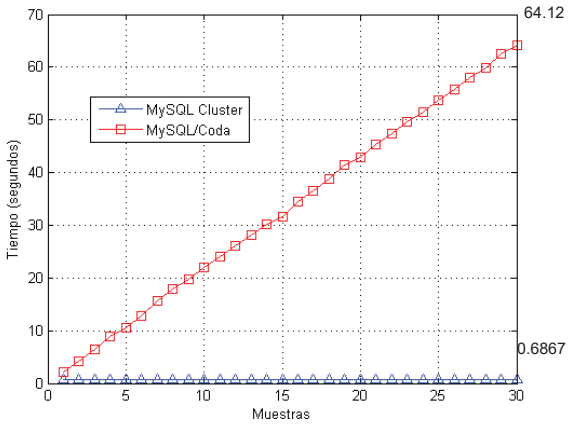


Fig. 5.4 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, consulta 3, todos los nodos activos.

En este caso, de los datos obtenidos mediante el método de aproximación de mínimos cuadrados se tiene que el valor de m (pendiente de la recta) para la curva de MySQL Cluster es un valor muy cercano a cero, lo que representa que la curva es una línea recta con la ecuación $y = 0.718$, por lo que el cálculo de la eficiencia mostraría un valor que tiende a ∞, que indica que el tiempo consumido por Coda también tendería al ∞. De los resultados anteriores, se puede observar que MySQL es más estable en su tiempo de respuesta y que por el contrario, para MySQL/Coda mientras el tamaño del problema aumenta, el tempo de respuesta tiende al ∞. En la Tabla 5.4 se muestran las ecuaciones de las rectas ajustadas.

Tabla 5.4 Ecuaciones de las rectas ajustadas consulta 3, todos los nodos activos. Caso de estudio 1.

	Ecuación de la recta
MySQL Cluster	$y = 0.63$
MySQL/Coda	$y = 2.14x + 0.218$

5.2.2 Caso de Estudio 2. Un nodo inactivo

En el segundo caso de estudio, para cada plataforma se simuló una falla de comunicación en un nodo que forma parte de la base de datos distribuida, dejándolo fuera tanto del cluster MySQL como de la célula Coda, obteniéndose los siguientes resultados:

En la Figura 5.5, se muestran los tiempos de respuesta en la inserción de registros a la base de datos, donde de acuerdo a los resultados obtenidos se puede verificar que MySQL/Coda es más eficiente que MySQL Cluster, en relación a los tiempos obtenidos por la plataforma clusterizada. Esta eficiencia alcanza su punto máximo en la prueba 30, donde MySQL/Coda es 3.82 veces más veloz que MySQL Cluster. También se puede observar que el tiempo requerido para hacer las operaciones de inserción se reduce para MySQL Cluster, con respecto a la medición tomada con todos los nodos activos y llega a ser hasta un 93.88% del tiempo tomado por todos los nodos, y para CODA hay una variación del tiempo respecto al registrado por todos los nodos de hasta ±8.60%. Debe tomarse en consideración que en ambos casos, se suprime la replicación a un nodo, debido a la falla.

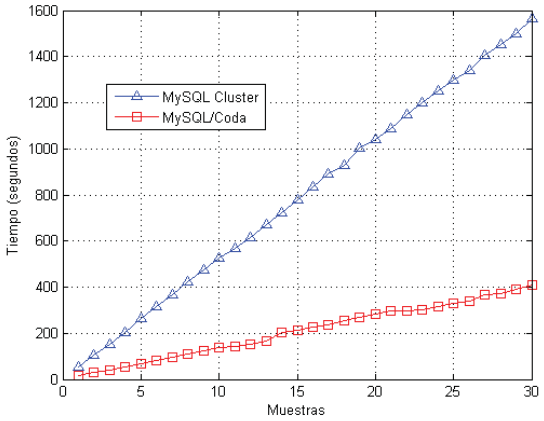


Fig. 5.5 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, inserción de registros, 1 nodos inactivo

Según los datos obtenidos analíticamente, la eficiencia mayor es presentada por la plataforma MySQL/Coda, en hasta 3.87 veces, coincidiendo con los resultados de los datos reales donde se expresó que esta plataforma es más eficiente hasta 3.866 veces. Las ecuaciones obtenidas por mínimos cuadrados se listan en la Tabla 5.5.

Tabla 5.5 Ecuaciones de las rectas ajustadas para inserción de registros, con un nodo inactivo. Caso de estudio 2.

	Ecuación de la recta
MySQL Cluster	$y = 51.904x + 1.946$
MySQL/Coda	$y = 13.424x + 2.574$

En la Figura 5.6 se muestran los tiempos de respuesta registrados durante la consulta 1 con un nodo inactivo en ambas plataformas. En este caso, el tiempo de respuesta de MySQL/Coda es más eficiente con respecto al de MySQL Cluster hasta en un 67.18%, en el último punto de las muestras de la prueba. Se registra para MySQL/Coda una variación en el tiempo de respuesta, con respecto al obtenido con todos los nodos activos, de hasta un ±4.63%; MySQL Cluster incrementa hasta un 9.31% su tiempo de respuesta.

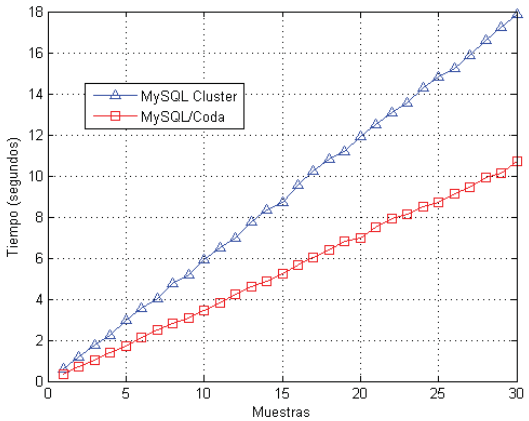


Fig. 5.6 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, consulta 1, 1 nodo inactivo

Los datos obtenidos de manera analítica expresan que la combinación MySQL/Coda llega a ser hasta 1.67 veces más eficiente que MySQL Cluster, dato que coincide con la eficiencia calculada con los datos reales. En la Tabla 5.6 se presentan las ecuaciones que predicen el comportamiento de las ambas plataformas en el tiempo.

Tabla 5.6 Ecuaciones de las rectas ajustadas para consulta 1, con un nodo inactivo. Caso de estudio 2.

	Ecuación de la recta
MySQL Cluster	$y = 0.594x - 0.342$
MySQL/Coda	$y = 0.354x - 0.018$

En la Figura 5.7, se muestran los tiempos de respuesta registrados durante la consulta 2 con un nodo inactivo en ambas plataformas; bajo estas condiciones MySQL Cluster es más eficiente que MySQL/Coda con respecto al tiempo de respuesta hasta 1.22 veces, en el último punto de las pruebas. Con respecto al tiempo de respuesta para la consulta 2 con todos los nodos activos, MySQL Cluster incrementa su tiempo hasta un 10.54% y MySQL/Coda varia en un rango de ±5.46%.

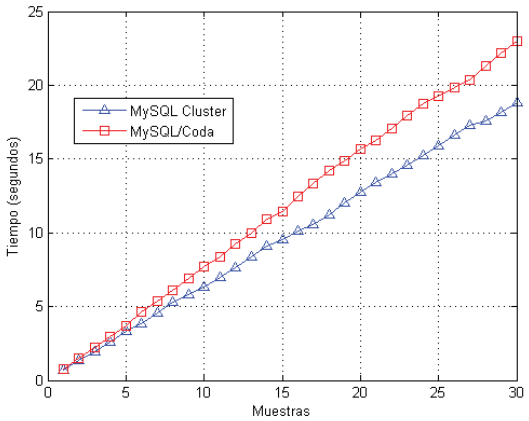


Fig. 5.7 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, consulta 2, 1 nodo inactivo

De la ecuación obtenida mediante el método de ajuste de mínimos cuadrados se obtuvo que MySQL Cluster es más eficiente que MySQL/Coda en hasta 1.225 veces en cuanto a tiempo de respuesta durante la ejecución de las pruebas de la consulta 2. Las ecuaciones del comportamiento de estas de las plataformas analizadas se muestran en la Tabla 5.7.

Tabla 5.7 Ecuaciones de las rectas ajustadas para consulta 2, con un nodo inactivo. Caso de estudio 2.

	Ecuación de la recta
MySQL Cluster	$y = 0.629x - 0.0342$
MySQL/Coda	$y = 0.77x - 0.00308$

En la Figura 5.8, se observa el tiempo de respuesta para realizar la consulta 3 para ambas plataformas. Se conserva el comportamiento observado en la prueba anterior, donde el tiempo de respuesta de MySQL Cluster para todas las pruebas se mantiene casi constante. Así, se registró que el tiempo de respuesta de MySQL Cluster es 84.48 veces más rápido que el de la plataforma MySQL/Coda en esta operación.

Con respecto al tiempo de respuesta para la consulta 3 con todos los nodos activos, MySQL Cluster incrementa su tiempo hasta un 15.61% y MySQL/Coda varia hasta un ±5.84%

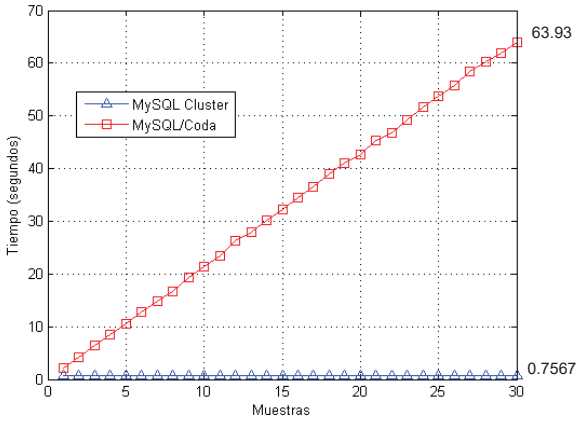


Fig. 5.8 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, consulta 3, 1 nodo inactivo.

Para esta prueba, se realizaron los cálculos para generar las ecuaciones de las rectas que expresan el comportamiento de ambas plataformas, observándose que MySQL Cluster se conserva con un valor casi constante, por lo que la pendiente de su ecuación es cercana a cero, lo cual representa una línea recta a través del tiempo; mientras que MySQL/Coda incrementa el tiempo de respuesta a esta consulta de una manera lineal y tiene a incrementar hasta el ∞. En la Tabla 5.8

Tabla 5.8 Ecuaciones de las rectas ajustadas para consulta 3, con un nodo inactivo. Caso de estudio 2.

	Ecuación de la recta
MySQL Cluster	$y = 0.64$
MySQL/Coda	$y = 2.146x - 0.0114$

5.2.3 Caso de Estudio 3. Dos nodos inactivos

En el tercer caso de estudio, para cada plataforma se simuló la falla de comunicación de dos nodos de la base de datos distribuida, dejándolos fuera de funcionamiento tanto del cluster MySQL como de la célula Coda, obteniéndose los resultados que a continuación se describen.

Se encontró que la plataforma MySQL/Coda es más eficiente que MySQL Cluster en relación a los tiempos de respuesta obtenidos en la inserción de registros a la base de datos durante la situación de falla mencionada anteriormente. Dicha eficiencia alcanza un 96.71% durante la prueba 30. También se observó que el tiempo para ejecutar las operaciones de inserción se reduce para la plataforma MySQL/Coda hasta en un 49.18% con respecto al tiempo requerido por esta plataforma para realizar tales operaciones incluyendo todos los nodos activos. En el caso de Coda, existe una variación del tiempo registrado para realizar la inserción de registros con 2 nodos inactivos, con respecto al requerido para realizar esta operación con todos los nodos activos de ±9.10%.

Debe tomarse en consideración que en ambos casos, se suprime la replicación en dos nodos, debido a la falla de comunicación. La Figura 5.9 muestra una comparación de los tiempos de respuesta obtenidos para ambas implementaciones.

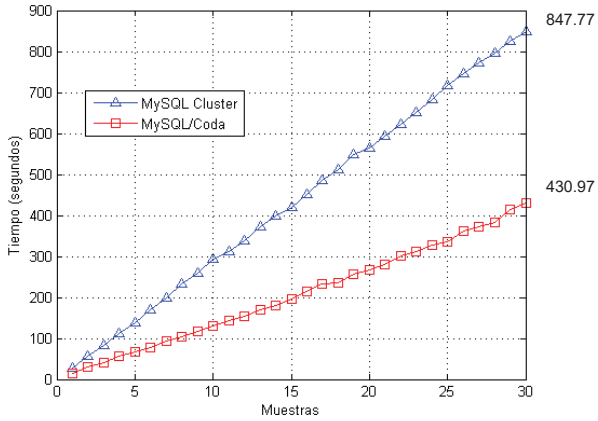


Fig. 5.9 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, con inserción de registros, y considerando 2 nodos inactivos

De los datos calculados mediante mínimos cuadrados se puede observar que MySQL/Coda es más eficiente que MySQL Cluster durante la inserción de registros hasta 2.046 veces, siendo esta eficiencia ligeramente mayor a la calculada con los datos reales de 1.96 veces. En la Tabla 5.9 se muestra las ecuaciones para cada las aplicaciones aquí propuestas.

Tabla 5.9 Ecuaciones de las rectas ajustadas para inserción de registros, con dos nodos inactivos. Caso de estudio 3.

	Ecuación de la recta
MySQL Cluster	$y = 28.467x - 0.0624$
MySQL/Coda	$y = 13.911x - 5.451$

Durante las pruebas de ejecución de la consulta 1 con dos nodos inactivos, la plataforma MySQL/Coda mostró tener mejores tiempos de respuesta en comparación con MySQL Cluster, logrando ser más eficiente hasta 1.68 veces en el punto 30 de las pruebas. En la Figura 5.10 se observan los resultados obtenidos por ambas plataformas. También se registró que MySQL Cluster llega a incrementar el tiempo de respuesta a la consulta 1 con respecto a los tiempos obtenidos cuando se encuentran todos los nodos activos de hasta un 14.77%, mientras que MySQL/Coda varia hasta en un 4.94% los tiempos de la consulta 1 con respecto a los obtenidos en las pruebas del caso de estudio 1.

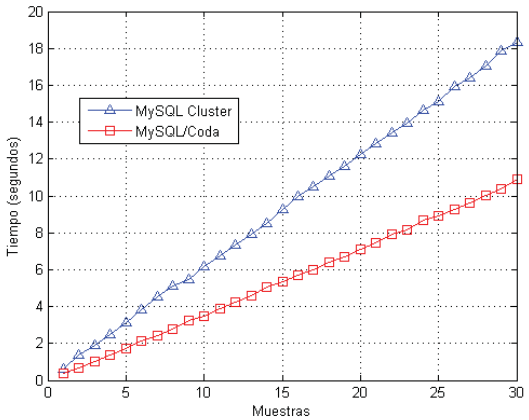


Fig. 5.10 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, consulta 1, 2 nodos inactivos

Para esta prueba, el resultado analítico, también favoreció MySQL Cluster, pues con los datos de de las ecuaciones encontradas, se determinó que la eficiencia de esta plataforma es 1.68 veces mayor en el tiempo de respuesta en las circunstancias de este caso de estudio. En la tabla 5.10 se muestran las ecuaciones de las líneas aproximadas por mínimos cuadrados.

Tabla 5.10 Ecuaciones de las rectas ajustadas para la consulta 1, con dos nodos inactivos. Caso de estudio 3.

	Ecuación de la recta
MySQL Cluster	$y = 0.606x + 0.1057$
MySQL/Coda	$y = 0.36x - 0.049$

En la Figura 5.11 se muestra los tiempos de respuesta obtenidos por ambas plataformas durante la ejecución de la consulta 2, donde se puede apreciar que MySQL Cluster logra obtener mejores tiempos de respuesta que la combinación de MySQL/Coda. La mejor eficiencia se obtuvo en la primera prueba, donde MySQL Cluster llega a ser 14.37 veces más rápido que MySQL/Coda para ejecutar la consulta 2. Los tiempos de respuesta obtenidos para la consulta 2 con 2 nodos inactivos por MySQL Cluster llegan a ser hasta un 27.64% mayores que los registrados con esta consulta con todos los nodos activos. En el caso de Coda, los tiempos obtenidos durante esta prueba llegan a ser mayores que los obtenidos con todos los nodos activos hasta 14.37 veces para la prueba número uno.

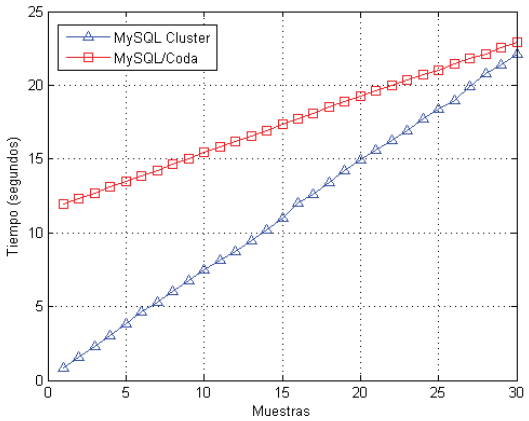


Fig. 5.11 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, consulta 2, 2 nodos inactivos

Para este caso particular, de la Figura 5.12 se puede visualizar que las curvas de ambas plataformas tienden a cruzarse en algún punto en el tiempo. Con las ecuaciones de ambas rectas se predijo el comportamiento de estas líneas.

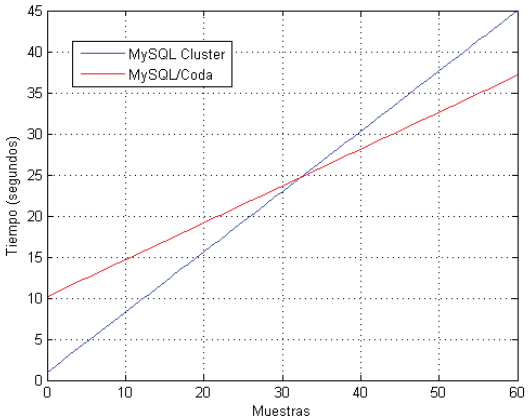


Fig. 5.12 Curvas ajustadas para MySQL Cluster y MySQL/Coda, consulta 2, 2 nodos inactivos

De los datos calculados con el ajuste de las líneas de estas plataformas para este caso, se obtuvo que existe una eficiencia relativa mayor de hasta 1.63 de MySQL/Coda con respecto a MySQL, lo cual indica que esta eficiencia se logra después del punto de intersección de ambas curvas. Puede observarse que mientras el tamaño del problema crece, la eficiencia y la rapidez de MySQL Cluster disminuyen. En la Tabla 5.11 se muestran las ecuaciones correspondientes a esta prueba.

Tabla 5.11 Ecuaciones de las rectas ajustadas para la consulta 2, con dos nodos inactivos. Caso de estudio 3.

	Ecuación de la recta
MySQL Cluster	$y = 0.734x + 0.956$
MySQL/Coda	$y = 0.449x + 10.1807$

En los tiempos de respuesta obtenidos durante las pruebas de ejecución de la consulta 3, los tiempos de respuesta para MySQL/Coda se incrementan de un forma casi lineal, mientras que para MySQL Cluster se mantienen en un rango de variación muy cerrado, siendo su comportamiento prácticamente constante, según se puede observar en la Figura 5.13. Dado este comportamiento, MySQL Cluster llega a ser hasta 83.31 veces más rápido que MySQL/Coda en el punto 30 de las pruebas. Con respecto a las pruebas realizadas en el caso 1, donde se consideran activos todos nodos; para MySQL Cluster las pruebas con 2 nodos inactivos llegan a tomar hasta un 21.46% más tiempo del registrado en el caso 1. MySQL/Coda varía los tiempos registrados con respecto a los obtenidos con todos los nodos activos en un ±3.58%.

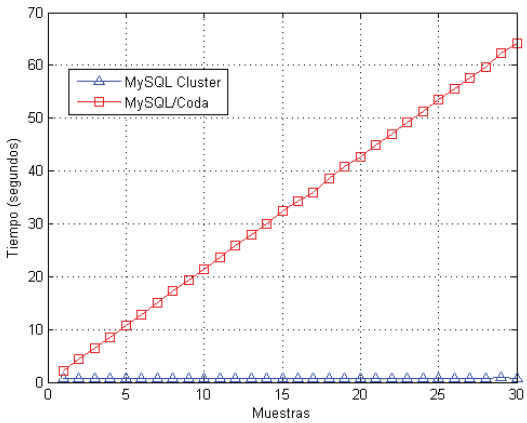


Fig. 5.13 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, consulta 3, 2 nodos inactivos

Para esta última prueba de la primera etapa, de manera analítica se observó que MySQL Cluster presenta un comportamiento que tiende a ser constante, representad por una línea recta con pendiente 0 y que MySQL/Coda incrementa de manera lineal su tiempo de respuesta así como aumenta el tamaño del problema a resolver. En la Tabla 5.12 se listan las ecuaciones de las rectas aproximadas para este caso.

Tabla 5.12 Ecuaciones de las rectas ajustadas para la consulta 3, con dos nodos inactivos. Caso de estudio 3.

	Ecuación de la recta
MySQL Cluster	$y = 0.697$
MySQL/Coda	$y = 2.135x + 0.08884$

5.2.4 Conclusiones

Durante esta primera etapa en los casos de estudio, se ha mostrado la operación de la base de datos distribuida sobre las dos plataformas propuestas, MySQL Cluster y la combinación de MySQL estándar con Coda sobre una primera configuración de equipo de cómputo, donde los nodos que manejan los datos distribuidos son computadoras personales con las misma características de hardware.

La operación de la base de datos distribuida ha tenido mayor eficiencia en cuanto tiempo de respuesta en las operaciones de inserción de registros sobre MySQL/Coda que la mostrada por MySQL Cluster en hasta 4.07 veces. La eficiencia en cuanto tiempo de respuesta también es mayor para MySQL/Coda en las consultas que requieren una recuperación completa de los datos de la tabla en hasta 1.56 veces en relación a MySQL Cluster. En las consultas, donde se incorpora el manejo de índices y se requiere la relación de dos tablas, MySQL Cluster llega a ser

notablemente más eficiente en hasta 1.33 veces en relación con el ambiente MySQL/Coda. Por otro lado, cuando al uso de índices se le incorpora un filtro como el WHERE, MySQL Cluster hacer uso de sus consultas paralelas y llega a superar a MySQL/Coda en eficiencia hasta 93.37 veces.

En la tabla 5.13 se muestra el número de veces que una de las plataformas propuestas es más eficiente que la otra.

Tabla 5.13 Número de veces que una plataforma es más eficiente que otra

Tipo de Operación	MySQL Cluster	MySQL/Coda
Inserción de registros	4.07	-
Consulta 1	1.56	-
Consulta 2	-	1.33
Consulta 3	-	93.37

5.3 Parte dos. 10 nodos de datos

La configuración realizada para MySQL Cluster consiste de 1 Nodo Administrador (MGM), dos nodos MySQL y diez nodos de datos heterogéneos distribuidos en cuatro computadoras, sobre los cuales se realizan las pruebas de eficiencia y disponibilidad, ver la Tabla 2.3 para consultar la distribución de los nodos de datos y la Tabla 2.4, pág. 18, donde se listan las características físicas de los nodos.

La configuración de MySQL/Coda esta conformada por un nodo servidor SCM, un nodo cliente y diez volúmenes de datos que son distribuidos en cuatro computadoras. Ver en la Tabla 2.6 las características del equipo de cómputo de esta configuración y la Tabla 2.7, pág 20, la distribución de los volúmenes de información.

La conexión de la red se hace mediante un switch Gigabit. Las fallas de los nodos fueron simuladas para el caso de MySQL Cluster deteniendo el demonio *ndbd*, del nodo de datos que se iba a presentar en falla y para el caso de MySQL/Coda el servidor que contenía los volúmenes reportados en falla se mantuvo apagado.

5.3.1 Caso de Estudio 4. Todos los nodos activos

En el cuarto caso de estudio el equipo considerado para realizar las pruebas de operación de la base de datos es heterogéneo, se considera que todos los nodos se encuentran activos y en la configuración se toma en cuenta que se realizan dos réplicas de cada fragmento de la base de datos de manera automática en ambas plataformas. Aplicando las operaciones citadas en el Capítulo 2, se obtuvieron los resultados y gráficas comparativas de desempeño entre MySQL Cluster y MySQL/Coda que se describen continuación.

En la Tabla 5.14 se presenta la evolución de los tiempos de respuesta en segundos a las operaciones de inserción a la base de datos, tanto para MySQL Cluster como para MySQL/Coda.

Tabla. 5.14 Tiempos de respuesta en las operaciones de inserción. Caso de estudio 4.

No. Prueba	No. Registros	MySQL Cluster	MySQL/Coda
1	66666	72.09	16.81
2	133332	144.49	30.56
29	1933314	2143.70	453.75
30	1999980	2201.65	465.35

En la Figura 5.14 se puede observar que el tiempo de respuesta en las operaciones de inserción a la base de datos distribuida son menores en la plataforma MySQL/Coda que los tiempos de respuesta obtenidos por MySQL Cluster, de manera similar con el caso de estudio 1. La eficiencia mayor pudo alcanzarse en la prueba número 30, donde a plataforma MySQL/Coda es 4.73 veces más rápida para realizar inserciones que MySQL Cluster.

Como en la primera etapa de los casos de estudio, se realizó un ajuste de curvas para cada una de las pruebas realizadas. Para la inserción de registros con todos los nodos activos y con nodos heterogéneos de datos se obtuvo que MySQL/Coda es más eficiente que MySQL en hasta 4.837 veces. En la Tabla 5.15 se presentan las ecuaciones de las curvas de los dos ambientes propuestos.

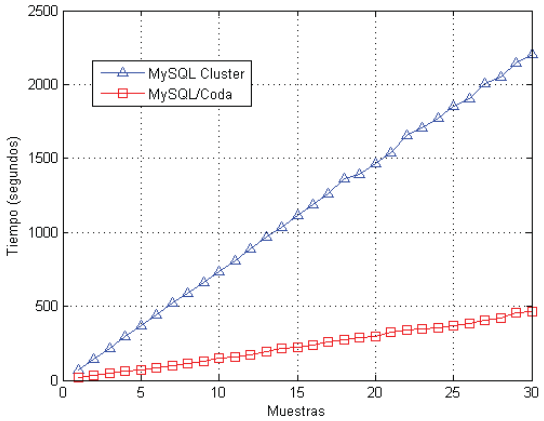


Fig. 5.14 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, inserción de registros, todos los nodos activos.

Tabla 5.15 Ecuaciones de las rectas ajustadas inserción de registros, con todos los nodos activos. Caso de estudio 4.

	Ecuación de la recta
MySQL Cluster	$y = 73.844x + 0.8265$
MySQL/Coda	$y = 15.265x - 3.973$

La siguiente prueba realizada en este caso de estudio consistió en ejecutar la consulta 1, donde se recuperan los datos de una manera secuencial y de acuerdo con el optimizador de MySQL se realiza un barrido completo de la tabla en cuestión; datos representativos de los tiempos de respuesta obtenidos durante esta consulta se muestran en la Tabla 5.16.

Tabla 5.16 Tabla comparativa de tiempos de respuesta durante la consulta 1 con todos los nodos activos. Caso de estudio 4.

No. Prueba	**No. Registros**	**MySQL Cluster**	**MySQL/Coda**
1	66666	0.44	0.36
2	133332	0.94	0.71
3	199998	1.39	1.06
15	999990	6.75	5.18
16	1066656	7.40	5.55
28	1866648	12.05	9.72
29	1933314	12.39	10.23
30	1999980	12.47	10.39

Es digno se notarse que durante este caso de estudio, los tiempos registrados por MySQL Cluster tuvieron un comportamiento más irregular, es decir, algunos puntos se alejan de la línea recta, aunque el comportamiento de esta consulta se mantiene casi lineal. Tal condición irregular en la linealidad es atribuida al método de recuperación de datos utilizado por MySQL Cluter, ya que en

este caso se utiliza el barrido completo de la tabla (*Full Table Scan*) [Davies y Fisk, 2006], el cual provoca que la tabla entera sea enviada a través de la red al servidor MySQL que la solicitó y en este caso los nodos de datos no se encuentran transmitiendo datos a una velocidad uniforme sobre la red. Dadas estas circunstancias, el mejor punto de eficiencia se encontró en la prueba número 16 donde MySQL/Coda es una plataforma 1.33 veces más rápida que MySQL Cluster para ejecutar la consulta 1, como se muestra en la Figura 5.15.

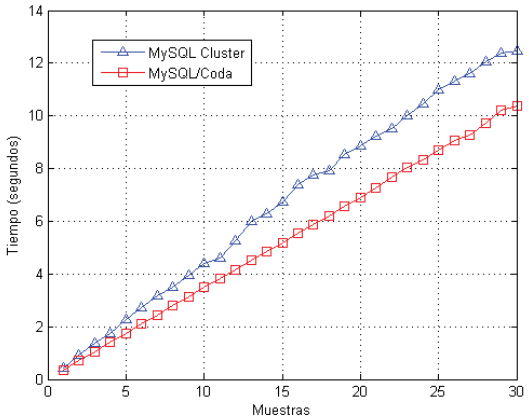


Fig. 5.15 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, consulta 1, todos los nodos activos.

De los datos obtenidos del ajuste de curvas se ratificó que la eficiencia de MySQL/Coda es mayor en esta prueba y según la ecuación de las líneas, esta eficiencia es mayor hasta 1.235 veces. La Tabla 5.17 muestra las ecuaciones para el comportamiento de MySQL Cluster y de MySQL/Coda.

Tabla 5.17 Ecuaciones de la consulta 1, con todos los nodos activos. Caso de estudio 4.

	Ecuación de la recta
MySQL Cluster	$y = 0.4288x + 0.1441$
MySQL/Coda	$y = 0.347x + 0.0053$

En la Tabla 5.18 se muestran algunos tiempos de respuesta representativos durante la ejecución de la consulta 2, considerando que se encuentran en operación todos los nodos de datos.

Tabla 5.18 Tabla comparativa de tiempos de respuesta durante la consulta 2 con todos los nodos activos. Caso de estudio 4.

No. Prueba	No. Registros	MySQL Cluster	MySQL/Coda
1	66666	0.63	0.73
2	133332	1.05	1.48
3	199998	1.48	2.23
12	799992	5.50	8.91
15	999990	6.95	11.19
17	1133322	7.99	12.69
30	1999980	13.76	22.25

Como se puede apreciar en la Figura 5.16, los tiempos de ejecución para realizar la consulta 2, son menores para MySQL Cluster que los registrados con la plataforma MySQL/Coda y calculando la eficiencia con los datos de la Tabla 5.18, se obtuvo que MySQL Cluster es hasta 1.62 veces más rápido que MySQL/Coda; de acuerdo con el optimizador de MySQL, MySQL Cluster empleó un método de acceso a los datos de índices ordenados y MySQL estándar usó una combinación de acceso de llave primaria y llave única.

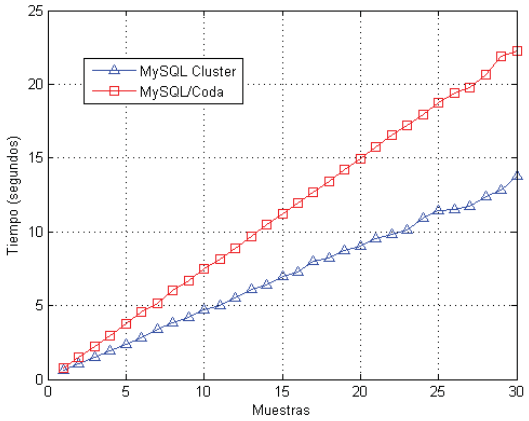


Fig. 5.16 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, consulta 2, todos los nodos activos.

De manera analítica, se encontró que bajo las últimas condiciones mencionadas, MySQL Cluster es más eficiente que MySQL/Coda hasta 1.689 veces. La Tabla 5.19 muestra las ecuaciones de ajuste a mínimos cuadrados obtenidas.

Tabla 5.19 Ecuaciones de la consulta 2, con todos los nodos activos. Caso de estudio 4.

	Ecuación de la recta
MySQL Cluster	$y = 0.4413x + 0.2078$
MySQL/Coda	$y = 0.7456x + 0.0102$

La Tabla 5.20 muestra algunos de los tiempos de respuesta obtenidos para la consulta 3, dentro de este caso de estudio, donde se consideran activos todos los nodos.

Tabla 5.20 Tabla comparativa de tiempos de respuesta durante la consulta 3 con todos los nodos activos. Caso de estudio 4.

No. Prueba	**No. Registros**	**MySQL Cluster**	**MySQL/Coda**
1	66666	1.10	2.23
2	133332	1.15	4.36
3	199998	1.12	6.46
14	933324	1.23	30.07
15	999990	1.15	32.08
16	1066656	1.17	34.31
28	1866648	1.20	58.25
29	1933314	1.21	62.29
30	1999980	1.18	63.01

En la Figura 5.17 se aprecia la gráfica de los tiempos de respuesta obtenidos durante la ejecución de la consulta 3 en ambas plataformas. Como puede observarse, el tiempo de respuesta para la plataforma MySQL/Coda crece de forma casi lineal a medida que el número de registros contenidos aumenta, mientras que para MySQL Cluster, el tiempo de respuesta permanece casi constante para todos las pruebas ejecutadas, sin exceder el limite de 1.5 seg. durante todas las pruebas. Para esta consulta el optimizador de MySQL ha activado la opción, *engine_condition_pushdown* [Davies y Fisk, 2006], la cual permite al servidor MySQL realizar un método de acceso a los datos más optimizado y la cláusula WHERE podrá ser procesada por cada nodo de datos antes de enviar el resultado de regreso al servidor MySQL. De esta forma, se obtuvo que MySQL Cluster es hasta 53.39 veces más rápido que MySQL/Coda en la consulta 3.

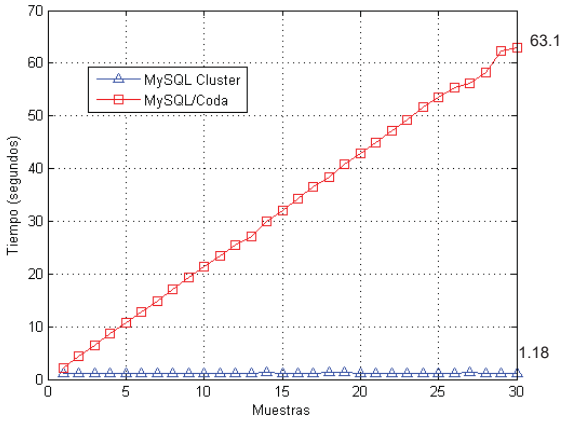


Fig. 5.17 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, consulta 3, todos los nodos activos.

De los datos generados con la aproximación de mínimos cuadrados se obtuvo que MySQL Cluster tiene un comportamiento cercano a lo constate, ya que la recta obtenida tiene una pendiente muy cercana a cero, mientras que el comportamiento de MySQL/Coda se describe como linealmente creciente. Las ecuaciones que describen a MySQL Cluster y a MySQL/Coda se presentan en la Tabla 5.21.

Tabla 5.21 Ecuaciones de la consulta 3, con todos los nodos activos. Caso de estudio 4.

	Ecuación de la recta
MySQL Cluster	$y = 0.995$
MySQL/Coda	$y = 2.1181x + 0.1774$

5.3.2 Caso de Estudio 5. Un nodo inactivo

En el quinto caso de estudio, dado de que alta disponibilidad es importante en un sistema de bases de datos distribuido, para cada plataforma se simuló una falla de comunicación en un nodo que forma parte de la base de datos distribuida, dejándolo fuera de operación tanto del cluster MySQL como de la célula Coda, obteniéndose los resultados que se describen a continuación.

En la Tabla 5.22 se presenta una muestra de los tiempos de respuesta a las operaciones de inserción a la base de datos tanto para MySQL Cluster como para MySQL/Coda.

Tabla 5.22 Tiempos de respuesta en las operaciones de inserción. Caso de estudio 5.

No. Prueba	No. Registros	MySQL Cluster	MySQL/Coda
1	66666	68.90	16.09
2	133332	140.42	30.54
3	199998	212.44	45.58
14	933324	975.09	207.17
28	1866648	1978.13	433.95
29	1933314	2036.15	448.13
30	1999980	2117.29	458.32

En la Figura 5.18 se muestra que el tiempo de respuesta en segundos de las operaciones de inserción a la base de datos distribuida cuando un nodo de datos se encuentra inactivo; se puede observar que los tiempos de la plataforma MySQL/Coda son menores que los obtenidos por MySQL Cluster; tomando como base los datos mostrados en la Tabla 5.22 se puede verificar que el punto de mayor eficiencia se alcanza en la prueba número 30, donde a plataforma MySQL/Coda es 4.62 veces más rápida para realizar inserciones que MySQL Cluster.

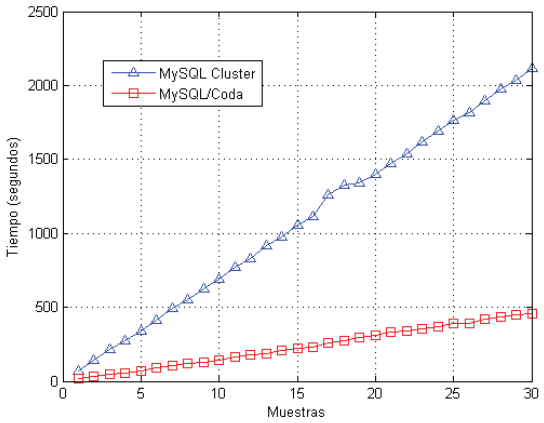


Fig. 5.18 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, inserción de registros, un nodo inactivo

Se observó que el tiempo requerido para realizar las operaciones de inserción a la base de datos se redujo en el caso de MySQL Cluster hasta un 96.17%, con respecto al tiempo registrado para hacer las inserciones cuando todos los nodos se encuentran activos; para MySQL/Coda existe una variación de ±8.25% realizando la comparación antes mencionada.

Analíticamente, se comprobó que la eficiencia de MySQL/Coda es mayor que la de MySQL Cluster en la inserción de registros con un nodo inactivo y de acuerdo a los datos calculados de esta forma, se determinó que esta eficiencia es mayor en hasta 4.546 veces. En la Tabla 5.23 se muestran las ecuaciones obtenidas con el método de regresión de mínimos cuadrados para las dos plataformas propuestas.

Tabla 5.23 Ecuaciones de la inserción de registros, con un nodo activo. Caso de estudio 5.

	Ecuación de la recta
MySQL Cluster	$y = 70.6635x - 3.9429$
MySQL/Coda	$y = 15.5436x - 4.6137$

A continuación, en la Tabla 5.24 se presenta una muestra de los tiempos de respuesta en segundos a la consulta 1 para MySQL Cluster como para MySQL/Coda.

En la Figura 5.19 se muestran los tiempos de respuesta registrados durante la consulta 1 con un nodo inactivo en ambas plataformas. En este caso, el tiempo de respuesta de MySQL/Coda es hasta 1.33 veces más rápido que el de MySQL Cluster en el último punto de las muestras de esta prueba. Se registra para MySQL/Coda una variación en el tiempo de respuesta, con respecto al obtenido con todos los nodos activos, de hasta un ±2.11%; MySQL Cluster incrementa hasta un 10.10% su tiempo de respuesta.

Tabla 5.24 Tiempos de respuesta en la consulta 1. Caso de estudio 5.

No. Prueba	No. Registros	MySQL Cluster	MySQL/Coda
1	66666	0.48	0.38
2	133332	0.91	0.71
3	199998	1.33	1.06
15	999990	6.61	5.18
17	1133322	7.54	5.88
28	1866648	12.53	9.66
29	1933314	13.05	10.16
30	1999980	13.73	10.36

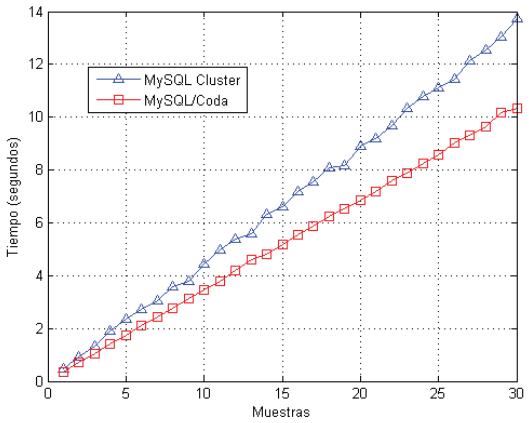


Fig. 5.19 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, consulta 1, un nodo inactivo

De acuerdo a los datos obtenidos por el método de ajuste de curvas por mínimos cuadrados se obtuvo que MySQL/Coda es más eficiente que MySQL Cluster hasta 1.298 veces. En la Tabla 5.25 se indican las ecuaciones que describen el comportamiento de ambas plataformas.

Tabla 5.25 Ecuaciones de la consulta 1, con un nodo activo. Caso de estudio 5.

	Ecuación de la recta
MySQL Cluster	$y = 0.4474x - 0.0234$
MySQL/Coda	$y = 0.3445x + 0.0185$

En la Tabla 5.26, se muestran algunos de los tiempos de respuesta registrados durante la consulta 2 con un nodo inactivo en ambas plataformas; bajo estas condiciones, como se puede observar en la Figura 5.20 el tiempo de respuesta de MySQL Cluster es menor con respecto al tiempo obtenido con MySQL/Coda hasta en 1.60 veces, en el último punto de las pruebas. Con respecto al tiempo de respuesta para la consulta 2 con todos los nodos activos, MySQL Cluster incrementa su tiempo hasta un 6.06% y MySQL/Coda varia en un rango de ±7.21%.

Tabla 5.26 Tiempos de respuesta en la consulta 2. Caso de estudio 5.

No. Prueba	**No. Registros**	**MySQL Cluster**	**MySQL/Coda**
1	66666	0.62	0.74
2	133332	1.07	1.59
3	199998	1.54	2.24
16	1066656	7.66	11.78
29	1933314	13.40	21.54
30	1999980	13.73	22.00

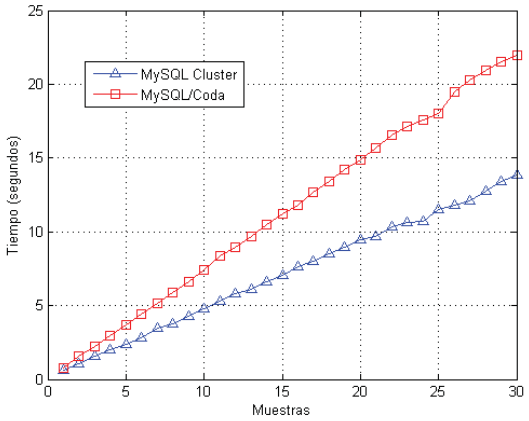


Fig. 5.20 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, consulta 2, un nodo inactivo

De las ecuaciones obtenidas por el método de mínimos cuadrados se obtuvo que MySQL Cluster es más eficiente que MySQL/Coda durante la consulta 2, hasta en 1.641 veces. La Tabla 5.27 muestra las ecuaciones ajustadas de los dos ambientes propuestos.

Tabla 5.27 Ecuaciones de la consulta 2, con un nodo activo. Caso de estudio 5.

	Ecuación de la recta
MySQL Cluster	$y = 0.4515x + 0.2259$
MySQL/Coda	$y = 0.7407x + 0.0392$

En la última prueba de este caso de estudio se obtuvieron los tiempos de respuesta para la consulta 3 sobre las dos plataformas propuestas, usando equipos heterogéneos. En la Tabla 5.28, se indican los tiempos de ejecución obtenidos.

Tabla 5.28 Tiempos de respuesta en la consulta 3. Caso de estudio 5.

No. Prueba	**No. Registros**	**MySQL Cluster**	**MySQL/Coda**
1	66666	1.09	2.23
2	133332	1.13	4.32
3	199998	1.11	6.41
16	1066656	1.16	33.48
28	1866648	1.17	59.55
29	1933314	1.32	61.58
30	1999980	1.11	62.86

Como puede apreciarse de la Figura 5.21, el tiempo de respuesta para realizar la consulta 3, conserva el comportamiento observado cuando se encuentran en funcionamientos todos los nodos que conforman la base de datos distribuida, donde el tiempo de respuesta de MySQL Cluster para todas las pruebas se mantiene prácticamente constante. El tiempo de respuesta de MySQL Cluster llega a ser hasta 56.63 veces más rápido que el de la plataforma MySQL/Coda en esta operación.

Con respecto al tiempo de respuesta para la consulta 3 con todos los nodos activos, MySQL Cluster incrementa su tiempo hasta un 18.87% y MySQL/Coda varia hasta un ±11.62%

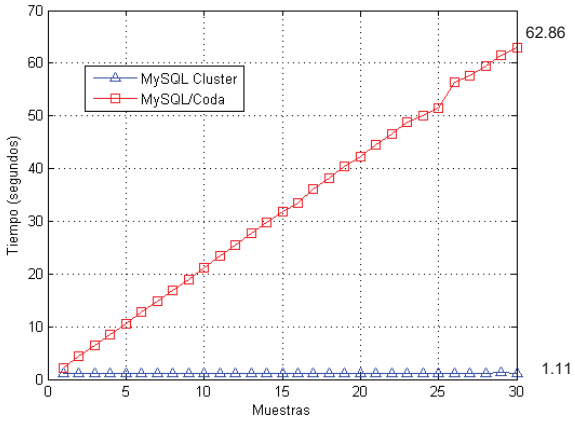


Fig. 5.21 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, consulta 3, un nodo inactivo

Según el comportamiento matemático de la plataforma MySQL Cluster se puede observar que sigue una trayectoria casi constante, ya que la pendiente de la recta encontrada tiende a ser cero, lo que refleja a una línea recta. Para MySQL/Coda el tiempo de respuesta aumenta conforme aumenta el tamaño del problema a resolver. En la Tabla 5.29 se muestran las ecuaciones obtenidas por mínimos cuadrados.

Tabla 5.29 Ecuaciones de la consulta 2, con un nodo activo. Caso de estudio 5.

	Ecuación de la recta
MySQL Cluster	$y = 0.9728$
MySQL/Coda	$y = 2.1143x + 0.0544$

5.3.3 Caso de Estudio 6. Cinco nodos inactivos

Para el último caso de estudio, se simuló la falla de comunicación del nodo más robusto que conforma el cluster heterogéneo, este servidor, además de poseer las mejores características de hardware, contiene el mayor número de nodos de datos en MySQL Cluster y el mayor número de volúmenes replicados en el sistema de archivos distribuido Coda. Estando sin funcionamiento los cinco nodos de datos de para el caso de MySQL Cluter y los cinco volúmenes de Coda, se obtuvieron los resultados que se describen a continuación.

Se registró que la plataforma MySQL/Coda es más eficiente que MySQL Cluster en relación a los tiempos de respuesta obtenidos en la inserción de registros a la base de datos durante la situación de falla de los 5 nodos de datos, como se puede apreciar en la Figura 5.22. Esta eficiencia, tiene punto máximo en la prueba de inserción 30, donde MySQL/Coda es más rápido que la plataforma clusterizada hasta en 5.44 veces, como se muestra en la Tabla 5.30, donde se encuentran algunos tiempos de respuesta de la consulta 3.

Tabla 5.30 Tiempos de respuesta en las operaciones de inserción. Caso de estudio 6.

No. Prueba	No. Registros	MySQL Cluster	MySQL/Coda
1	66666	82.00	17.29
28	1866648	2417.22	452.16
29	1933314	2465.74	457.02
30	1999980	2510.42	460.68

También se observó que el tiempo para ejecutar las operaciones de inserción se llega a incrementar para la plataforma MySQL/Coda hasta en un 20.56% con respecto al tiempo requerido por esta plataforma para realizar estas operaciones cuando todos los nodos se encuentran activos.

En el caso de Coda, existe una variación del tiempo registrado para realizar la inserción de registros con 5 nodos inactivos, con respecto al requerido para realizar esta operación, con todos los nodos activos de ±11.62%.

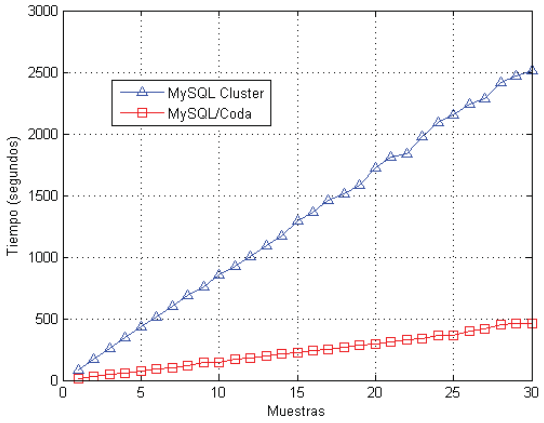


Fig. 5.22 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, inserción de registros, cinco nodos inactivos

Bajo las condiciones descritas, se generaron las ecuaciones de ajuste de curvas de mínimos cuadrados para MySQL Cluster y para MySQL/Coda y se verificó que MySQL/Coda es más eficiente que MySQL Cluster en las inserciones de registros con dos nodos inactivos en 5.554 veces. La Tabla 5.31 se muestran las ecuaciones de aproximación de ambas plataformas.

Tabla 5.31 Ecuaciones de inserción de registros, con cinco nodos inactivos. Caso de estudio 6.

	Ecuación de la recta
MySQL Cluster	$y = 85.3196x - 0.8045$
MySQL/Coda	$y = 15.3599x - 3.7739$

La siguiente prueba realizada en este caso de estudio, consistió en registrar los tiempos de respuesta en la ejecución de la consulta 1. La plataforma MySQL/Coda mostró tener mejores tiempos de respuesta en comparación con MySQL Cluster, logrando ser hasta 2.44 veces más rápido en su punto 1, como se puede observar de en la Tabla 5.32, donde se recopilaron algunos valores representativos de esta prueba.

Tabla 5.32 Tiempos de respuesta en la consulta 1. Caso de estudio 6.

No. Prueba	**No. Registros**	**MySQL Cluster**	**MySQL/Coda**
1	66666	0.89	0.36
2	133332	1.64	0.71
3	199998	2.51	1.06
14	933324	9.76	4.84
15	999990	10.63	5.16
16	1066656	11.25	5.52
28	1866648	19.73	9.71
29	1933314	20.18	9.99
30	1999980	20.56	10.30

En la Figura 5.23 se observan los resultados obtenidos por ambas plataformas. También se registró que MySQL Cluster llega a incrementar el tiempo de respuesta a la consulta 1 con respecto a los tiempos obtenidos con todos los nodos activos de hasta un 101.52%, mientras que

MySQL/Coda varia hasta en un 2.41% los tiempos de la consulta 1 con respecto a los obtenidos con todos los nodos activos.

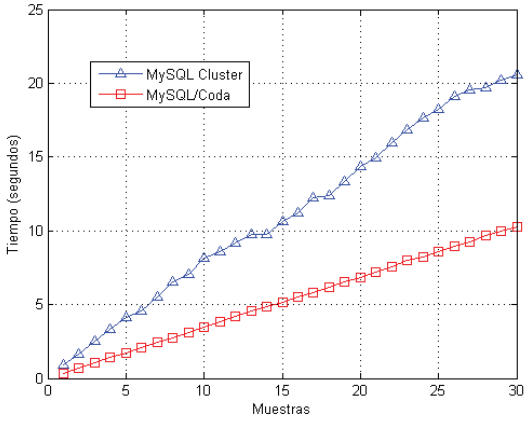


Fig. 5.23 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, consulta 1, cinco nodos inactivos

De los datos contenidos en las ecuaciones de ajuste de curvas de mínimos cuadrados para ambas plataformas se obtuvo que MySQL/Coda es más eficiente en esta prueba que MySQL Cluster hasta 2.02 veces. En la Tabla 5.33 se muestran las ecuaciones obtenidas para ambos ambientes.

Tabla 5.33 Ecuaciones de la consulta 1, con cinco nodos inactivos. Caso de estudio 6.

	Ecuación de la recta
MySQL Cluster	$y = 0.6934x + 0.5159$
MySQL/Coda	$y = 0.3432x + 0.0343$

A continuación se presentan los resultados obtenidos en la siguiente prueba que incluye el caso 6. En la Tabla 5.34 se presentan algunos de los tiempos de respuesta obtenidos en las pruebas de ejecución de la consulta 2, con cinco nodos en falla

Tabla 5.34 Tiempos de respuesta en la consulta 2. Caso de estudio 6.

No. Prueba	No. Registros	MySQL Cluster	MySQL/Coda
1	66666	1.19	0.75
2	133332	1.90	1.49
3	199998	2.84	2.31
14	933324	12.32	10.48
15	999990	12.63	11.17
16	1066656	12.77	11.94
28	1866648	23.14	20.93
29	1933314	24.26	21.67
30	1999980	25.87	22.63

En la Figura 5.24 se muestra la gráfica de los tiempos de respuesta obtenidos por ambas plataformas durante la ejecución de la consulta 2 con cinco nodos en falla. Se puede apreciar que MySQL Cluster logra obtener mejores tiempos de respuesta que la combinación de MySQL/Coda. El punto de máxima eficiencia se obtuvo en la primera prueba, donde MySQL Cluster llega a ser 1.58 veces más rápido que MySQL/Coda para ejecutar la consulta 2. Los tiempos de respuesta obtenidos para la consulta 2 con 5 nodos inactivos llegan a ser hasta un 100.70% mayores que los registrados con esta consulta con todos los nodos activos. En el caso de Coda, los tiempos obtenidos durante esta prueba, se encuentran variando en un ±3.44%.

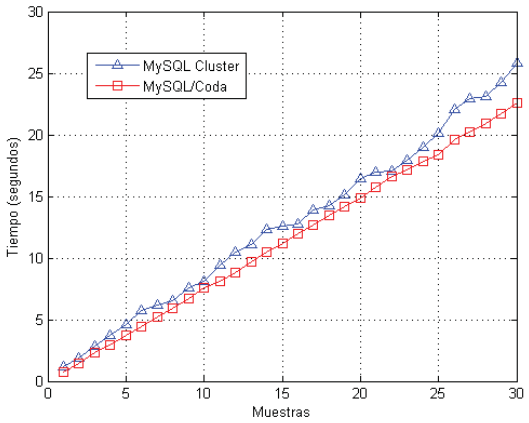


Fig. 5.24 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, consulta 2, cinco nodos inactivos

Se obtuvieron las ecuaciones que describen el comportamiento de ambas plataformas en las condiciones mencionadas, las cuales se muestran en la Tabla 5.35. De acuerdo a estas ecuaciones, MySQL Cluster reporta una eficiencia ligeramente mayor a la de MySQL/Coda en todas las ejecuciones que se proyecten. MySQL Cluster puede ser hasta 1.08 veces más rápido que MySQL/Coda.

Tabla 5.35 Ecuaciones de la consulta 2, con cinco nodos inactivos. Caso de estudio 6.

	Ecuación de la recta
MySQL Cluster	$y = 0.8111x + 0.2884$
MySQL/Coda	$y = 0.7485 - 0.0199$

Finalmente, como se puede observar en la Tabla 5.36, se listan los tiempos de respuesta obtenidos durante las pruebas de ejecución de la consulta 3, obsérvese que estos se incrementan de una manera casi lineal para MySQL/Coda, mientras que para MySQL Cluster se mantienen en un rango de variación muy cerrado, donde su comportamiento es prácticamente constante, según se puede observar en la Figura 5.25. De acuerdo a este comportamiento, MySQL Cluster llega a ser hasta 33.24 veces más rápido que MySQL/Coda en el punto 30 de las pruebas. Con respecto a las pruebas realizadas donde se consideran activos todos nodos; para MySQL Cluster las pruebas con 5 nodos inactivos llegan a tomar hasta un 75.23% más del tiempo registrado con todos los nodos. MySQL/Coda varia los tiempos registrados con respecto a los obtenidos con todos los nodos activos en un ±2.79%.

Tabla 5.36 Tiempos de respuesta en la consulta 3. Caso de estudio 6.

No. Prueba	No. Registros	MySQL Cluster	MySQL/Coda
1	66666	1.64	2.24
2	133332	1.75	4.36
3	199998	1.78	6.42
16	1066656	1.73	33.96
17	1133322	1.87	36.00
18	1199988	1.86	38.36
28	1866648	1.80	59.11
29	1933314	1.89	61.34
30	1999980	1.91	63.59

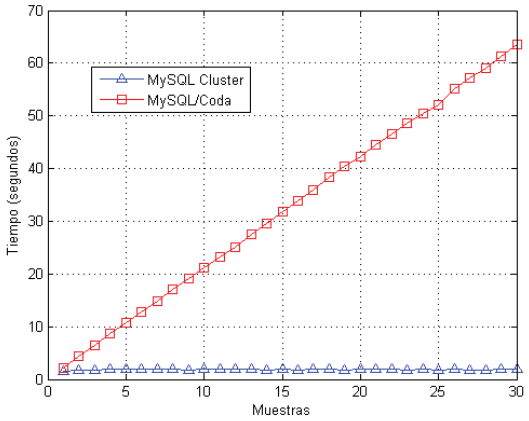


Fig. 5.25 Gráfica comparativa entre MySQL Cluster y MySQL/Coda, consulta 3, cinco nodos inactivos

En este último caso de estudio, el comportamiento de MySQL Cluster se mantiene presentando un comportamiento muy apegado a una línea recta, lo cual es soportado por los resultados de ajuste de curvas realizado donde la pendiente de la recta es muy cercana a 0. Por otro lado, MySQL/Coda los tiempos de respuesta incrementan considerablemente al incrementar el número de registros por procesar. La Tabla 5.37 lista las ecuaciones correspondientes a esta prueba.

Tabla 5.37 Ecuaciones de la consulta 3, con cinco nodos inactivos. Caso de estudio 6.

	Ecuación de la recta
MySQL Cluster	$y = 1.6154$
MySQL/Coda	$y = 2.1091x + 0.126$

5.3.4 Conclusiones

Se ha mostrado el manejo y la operación de una base de datos distribuida en dos diferentes configuraciones de equipo de cómputo. Se ha observado la capacidad de una aplicación como MySQL Cluster para el manejo de múltiples nodos en un mismo equipo servidor haciendo uso de las ventajas que proporcionan múltiples procesadores y una cantidad adecuada de memoria para una aplicación como MySQL Cluster que basa el manejo de su base de datos en memoria.

Se observó que la plataforma MySQL/Coda muestra mayor eficiencia que MySQL Cluster en las operaciones de inserción de registros durante la operación normal de una base de datos, realizándose éstas operaciones hasta 4.73 veces más rápido. La plataforma constituida por MySQL y Coda se sigue mostrando más eficiente en las operaciones que requieren barrido completo de la tabla, tal es el caso de la consulta 1, donde se obtuvo que MySQL/Coda en más rápido en sus tiempos de respuesta hasta 1.33 veces que MySQL Cluster durante la operación normal de la base de datos. Se observó qu este comportamiento más eficiente se conserva también durante las fallas de nodos.

Por otro lado, MySQL Cluster se muestra más eficiente en el manejo de operaciones que implican el acceso a datos mediante de índices, tal es el caso de la consulta 2, donde los tiempos de respuesta fueron hasta 1.62 veces más rápidos que MySQL/Coda durante la operación normal de la base de datos. Se observó que MySQL Cluster es considerablemente más rápido que MySQL/Coda en las operaciones que requieren acceso a datos a través de índices y con un filtro WHERE; de esta forma, MySQL Cluster llega a ser hasta 53.39 veces más rápido que MySQL/Coda en las operaciones que requieren manejo de índices, esta eficiencia se conserva si se presentan fallas en los nodos que conforman la base de datos.

A continuación se presenta la Tabla 5.38 donde se condensan los datos de eficiencia en cuanto a tiempo de respuesta de una plataforma respecto de otra, mostrándose el número de veces que una plataforma es más eficiente con respecto de la otra.

Tabla 5.38 Número de veces que una plataforma es más eficiente que otra

Tipo de Operación	MySQL Cluster	MySQL/Coda
Inserción de registros	4.73	-
Consulta 1	1.33	-
Consulta 2	-	1.62
Consulta 3	-	53.39

Capítulo 6
Conclusiones y Recomendaciones para Trabajos de Investigación Futuros

6.1 Conclusiones

Esta Tesis ha descrito el diseño, construcción y operación de un sistema de cómputo distribuido (cluster), basado en computadoras personales, haciendo uso de software de código libre.

También en esta Tesis se ha comparado el rendimiento de una base de datos distribuida implementada en dos plataformas diferentes, analizando el tiempo de respuesta y la disponibilidad en diferentes pruebas.

En las pruebas realizadas durante la primera etapa empleando 4 nodos de datos, se obtuvo una mayor eficiencia del tiempo de respuesta en las operaciones de inserción de registros sobre la plataforma MySQL/Coda en relación con MySQL Cluster, siendo MySQL/Coda hasta un 407.27% más eficiente. La eficiencia mencionada se conserva superior para MySQL/Coda en las operaciones de barrido completo de una tabla hasta en un 56.32%. Sin embargo, cuando el método de acceso hace uso de índices, MySQL Cluster supera a MySQL/Coda, hasta en un 33.19%. En las consultas con métodos de acceso con índices y un filtro WHERE, donde el optimizador de MySQL Cluster puede aplicar un filtro a los datos antes de enviar la respuesta a la consulta paralela ejecutada, reduciendo la cantidad de datos enviado a través de la red de manera muy considerable, lo que permite a MySQL Cluster presentar tiempos de respuesta cortos y constantes ante cualquier número de registros que conformen la base de datos; por lo que, MySQL Cluster llega ser hasta 93.37 veces más eficiente que MySQL/Coda.

En la segunda etapa de estos casos de estudio se usó equipo completamente heterogéneo y la base de datos estaba constituida por 10 nodos de datos. Se registró nuevamente que la plataforma MySQL/Coda es más eficiente que MySQL Cluster en las operaciones de inserción de registros en la operación normal de una base de datos, siendo hasta 4.73 veces más eficiente. En las operaciones que requieren un barrido completo de una tabla son realizadas de una manera más eficiente por MySQL/Coda hasta en 1.33 veces que los resultados obtenidos con MySQL Cluster. Aún con nodos fuera de operación el comportamiento antes mencionado se sigue conservando.

Sin embargo, también durante esta segunda etapa, MySQL Cluster es más eficiente en las consultas que acceden a los datos mediante índices, tal es el caso de la consulta 2, donde los tiempos de respuesta fueron hasta 1.62 veces más eficientes que los presentados por MySQL/Coda durante la operación normal de la base de datos. También se observó que, como en la primera parte de las pruebas MySQL Cluster conserva su comportamiento con tendencia a presentar tiempos de respuesta constantes aún si el tamaño de la base de datos crece significativamente. Para este caso, se encontró que MySQL Cluster es hasta 53.39 veces más eficiente que MySQL/Coda.

La plataforma construida con MySQL/Coda ha obtenido tiempos de respuesta eficientes en operaciones que se llevan a cabo secuencialmente, por lo que las aplicaciones que requieren la repetición de transacciones, como inserciones a la base de datos o consultas que ejecutan un barrido completo de la base de datos, además de disponibilidad y un ambiente distribuido pueden recurrir a su implementación sobre esta plataforma.

MySQL Cluster es una plataforma que proporciona alta disponibilidad, eficientes métodos de acceso a los datos y la implementación de un ambiente distribuido de manera sencilla,

proporcionando un ambiente adecuado a las aplicaciones que requieren un gran acceso a la información por medio de consultas indexadas.

Se realizó el modelo matemático de los casos de estudio presentados, con lo cual se mostró la relación de eficiencia que existe entre las dos plataformas propuestas en cada una de las pruebas presentadas. Así como el empleo de este modelo matemático para predecir el comportamiento de los sistemas a través del tiempo.

Se realizó la configuración de varios nodos de datos dentro de una misma computadora, para el caso de MySQL Cluster y de varios volúmenes de información en una sola computadora para el caso de MySQL/Coda, lo que permitió hacer uso de las capacidades de hardware del equipo de cómputo como multiprocesador, memoria y almacenamiento en disco duro.

Durante la segunda parte de los casos de estudio, se configuró equipo con características multi-core para alojar parte de la base de datos distribuida, observándose que el uso de esta tecnología en una operación de la base de datos distribuida no necesariamente representa un incremento en la eficiencia con que se realizan las operaciones ordinarias sobre la base de datos.

Debe tomarse en consideración que hay factores que afectan el rendimiento de la base de datos, tales como que la tecnología multi-core comparte el mismo bus y ancho de banda, lo que genera una amplia competencia por estos recursos. MySQL Cluster maneja protocolos internos para garantizar la integridad de la información (protocolo de compromiso de dos fases y el protocolo Heartbeat) y los protocolos que hacen posible la alta disponibilidad de la información, como el de replicación de datos, cada uno de estos procesos hace que la latencia de red se incremente y se vea reflejado en un decremento de la eficiencia en tiempo de respuesta proporcionado por el manejador de bases de datos distribuidas. El volumen de la información es otro factor que altera el rendimiento de la base de datos, pues mientras más información existe mayor intercambio de mensajes se realiza entre los nodos que conforman el sistema distribuido.

Durante la primera parte de los casos de estudio se realizó una implementación de una base de datos distribuida sobre equipos adquiridos a un bajo costo y de características físicas estándar, mostrándose la posibilidad de obtener resultados eficientes usando software libre y equipo de cómputo sin características especiales.

La implementación de una base de datos distribuida en MySQL Cluster es sencilla, ya que proporciona un ambiente estable, de alta confiabilidad y disponibilidad de la información, sacrificando por esta causa, en muchas ocasiones, el rendimiento de la base de datos.

Coda como un sistema distribuido de archivos proporciona uniformidad en la manera que el usuario visualiza un sistema de archivos, permitiendo que una estructura remota, sea considerada local y proporciona alta disponibilidad de los datos, sin embargo, las operaciones que implican índices, tan comunes dentro de las aplicaciones de base de datos presentan una baja eficiencia con respecto a MySQL Cluster.

En general, la elección de una base de datos distribuida de código abierto como MySQL Cluster es una alternativa factible cuando se planea desarrollar una aplicación, ya que es una plataforma robusta, diseñada para proporcionar al usuario alta disponibilidad de los datos, aunque en algunos tipos de operaciones se sacrifique rendimiento por disponibilidad.

MySQL Cluster hace un manejo eficiente de consultas que involucran índices y combinaciones de tablas, las cuales se presentan cotidianamente en la operación diaria de una base de datos distribuida, existiendo la posibilidad de que algunas transacciones secuenciales donde no se desempeña de manera tan eficiente sean ejecutadas mediante una solución administrativa, como la programación de dichas operaciones en un horario fuera del laboral o el uso de procesos por lotes.

El costo de desarrollar una aplicación de bases de datos distribuidas en MySQL Cluster es bajo en comparación con bases de datos por las que se paga por una licencia como Oracle®. El software licenciado tiene a su favor la gran existencia de documentación para sus productos, sin embargo, la capacitación y el mantenimiento de estos sistemas son rubros que también tienen un costo alto. Los sistemas de código abierto han ganado terreno y se puede encontrar documentación en la Internet, además de que brindan una oportunidad de aprendizaje para los desarrolladores ya que se cuenta con el código fuente y puede éste puede ser modificado libremente.

Es conveniente analizar las ventajas y desventajas que presenta un ambiente en el que se alojará una base de datos distribuida antes de realizar su instalación y configuración para asegurar el éxito de una implementación de base de datos distribuida. Conocer si un manejador de bases de datos proporciona características de alta disponibilidad, en qué tipo de arquitectura computacional puede configurarse, si brinda soporte algún tipo de soporte técnico y el costo de implementación, son parámetros que deben saberse para elegir el manejador de bases de datos adecuado a cada implementación.

6.2 Recomendaciones para trabajos futuros

Tomando como referencia la investigación reportada en esta tesis, se recomienda los siguientes trabajos de investigación relacionados con los temas tratados en esta tesis:

- Aplicación de la técnica de tiempo real a la implementación de bases de datos distribuidas.
- La implementación de bases de datos distribuidas en un ambiente heterogéneo de sistemas operativos.
- La implementación de bases de datos distribuidas en un ambiente heterogéneo de sistemas distribuidos de archivos.
- Le extensión del sistema de cómputo distribuido a un mayor número de nodos para observar el impacto en su eficiencia.
- Aplicar el cluster a diversas aplicaciones científicas y tecnológicas en los campos de sistemas computacionales, ingeniería, medicina, bioingeniería, etc., en donde es altamente deseable el procesamiento distribuido de tareas considerando su operación en tiempo real.

Anexo A

Instalación del Servidor SCM Coda

Para instalar tanto servidores como clientes de Coda es necesario obtener el código fuente de esta aplicación de su sitio Web, http://www.coda.cs.cmu.edu; para lo cual se debe modificar el archivo sources.list que se encuentra en la ruta /etc/apt (para la versión de Ubuntu 7.04 manejada). Como se muestra en la Tabla siguiente:

```
# cd /etc/apt
# vim sources.list

Agregando a sources:

## coda source packages
deb http://www.coda.cs.cmu.edu/debian stable/
```

Se actualiza la lista de paquetes con los lugares indicados por /etc/apt/sources.list

```
# apt-get update
```

Enseguida se instalan los paquetes necesarios para el servidor de Coda

```
# apt-get install coda-server
```

Y se ejecuta la instalación de la aplicación vice-setup, en donde se elige si se realizará una instalación para un servidor Coda SCM o no.

```
# vice-setup
```

Enseguida se muestra la historia de la instalación realizada sobre el servidor SCM durante la segunda etapa de pruebas:

```
# vice-setup
Welcome to the Coda Server Setup script!

You already have a file /etc/coda/server.conf!
Continueing will remove that file.
Do you want to continue? [yes/no] yes
Setting up config files for a coda server.
Do you want the file /etc/coda/server.conf created? [yes] yes
What is the root directory for your coda server(s)? [/vice] /vice
Setting up /vice.
Directories under /vice are set up.

Is this the master server, aka the SCM machine? (y/n) y

Setting up tokens for authentication.
The following token must be identical on all servers.
Enter a random token for update authentication : elefante
The following token must be identical on all servers.
Enter a random token for auth2 authentication : elefantt
The following token must be identical on all servers.
Enter a random token for volutil authentication : elefannt
tokens done!

Setting up the file list for update client
Filelist for update ready.
```

```
Now installing files specific to the SCM...

Setting up servers file.
Enter an id for the SCM server. (hostname vmedina-laptop)
The serverid is a unique number between 0 and 255.
You should avoid 0, 127, and 255.
serverid: 1
done!
Setting up users and groups for Coda

You need to give me a uid (not 0 or 1) and username (not root)
for a Coda System:Administrator member on this server,
(sort of a Coda super user)

I will create the initial administrative user with Coda password
"changeme". This user/password is only for authenticating with
Coda and not for logging into your system (i.e. we don't use
/etc/passwd authentication for Coda)

Enter the uid of this user: 1001
Enter the username of this user: admin

A server needs a small log file or disk partition, preferrably on a
disk by itself. It also needs a metadata file or partition of approx
4% of your filespace.

Raw partitions have advantages because we can write to the disk
faster, but we have to load a copy of the complete RVM data
partition into memory. With files we can use a private mmap, which
reduces memory pressure and speeds up server startup by several
orders of magnitude.

Servers with a smaller dataset but heavy write activity will
probably benefit from partitions. Mostly read-only servers with a
large dataset will definitely benefit from an RVM data file. Nobody
has really measured where the breakeven point is, so I cannot
really give any hard numbers.

------------------------------------------------------
WARNING: you are going to play with your partitions now.
verify all answers you give.
------------------------------------------------------

WARNING: these choices are not easy to change once you are up and running.

Are you ready to msset up RVM? [yes/no] yes

What will be your log file (or partition)? /var/tmp/log.raw

The log size must be smaller than the available space in the log
partition. A smaller log will be quicker to commit, but the log
needs to be large enough to handle the largest transaction. A
larger log also allows for better optimizations. We recommend
to keep the log under 30M log size, many people have successfully
used as little as 2M, and 20M has worked well with our servers.
What is your log size? (enter as e.g. '20M') 20M

Where is your data file (or partition)? /var/tmp/data.raw

The amount of RVM we need to store the metadata for a given
amount file space can vary enormously. If your typical data set
consists of many small files, you definitely need more RVM, but
if you tend to store large files (mp3s, videos or image data)
we don't need all that much RVM.

Here are some random samples,
  mp3 files     ~0.08MB RVM per GB.
  jpeg images   ~0.50MB RVM per GB.
  email folders ~37.8MB RVM per GB (maildir, 1 file per message)
  netbsd-pkgsrc  ~180MB RVM per GB (large tree but not much data)
```

```
To get a more precize number for your dataset there is a small
tool (rvmsizer) which can reasonably predict the amount of RVM
data we need for a file tree.

Remember that RVM data will have to be mmapped or loaded
into memory, so if anything fails with an error like
RVM_EINTERNAL you might have to add more swap space.

What is the size of you data file (or partition)
[32M, 64M, 128M, 256M, 512M, 768M, 1G]: 128M

------------------------------------------------------
WARNING: DATA and LOG partitions are about to be wiped.
------------------------------------------------------

  --- log area: /var/tmp/log.raw, size 20M.
  --- data area: /var/tmp/data.raw, size 128 MB.

Proceed, and wipe out old data? [y/n] y


LOG file has been initialized!


Rdsinit will initialize data and log.
This takes a while.
rvm_initialize succeeded.
Going to initialize data file to zero, could take awhile.
done.
rds_zap_heap completed successfully.
rvm_terminate succeeded.

RVM setup is done!


Directories on the server will be used to store container files
that hold the actual data of files stored in Coda. Directory
contents as well as metadata will be stored in the RVM segment
that we already configured earlier.

You should only have one container file hierarchy for each disk
partition, otherwise the server will generate incorrect
estimates about the actual amount of exportable disk space.

Where shall we store your file data [/vicepa]?
Shall I set up a vicetab entry for /vicepa (y/n) y
Select the maximum number of files for the server.
[256K, 1M, 2M, 16M]:
16M

Server directory /vicepa is set up!

Congratulations: your configuration is ready...

Shall I try to get things started? (y/n) y
 - Coda authentication server (auth2 &)
 - Coda update server (updatesrv)
 - Coda update client (updateclnt -h vmedina-laptop)
Creating /vice/spool
 - Coda file server (startserver)


Nice, it looks like everything went ok
Now I'll try to create an initial root volume
 - createvol_rep / vmedina-laptop/vicepa
Replicated volumeid is 7f000000
creating volume /.0 on vmedina-laptop (partition /vicepa)
V_BindToServer: binding to host vmedina-laptop
V_BindToServer: binding to host vmedina-laptop
Set Log parameters
Fetching volume lists from servers:
```

```
V_BindToServer: binding to host vmedina-laptop
GetVolumeList finished successfully
 vmedina-laptop - success
V_BindToServer: binding to host vmedina-laptop
VLDB completed.
<echo / 7f000000 1 01000001 0 0 0 0 0 0 0 >> /vice/db/VRList.new>
V_BindToServer: binding to host vmedina-laptop
VRDB completed.
Do you wish this volume to be Backed Up (y/n)? [n]

That seems to have worked...
If you have a working Coda client you should now be able to
access the new Coda realm
 - cfs lv /coda/vmedina-laptop/

enjoy Coda.
 for more information see http://www.coda.cs.cmu.edu.
```

Instalación del MySQL 5.1

El código fuente de MySQL 5.1 puede ser descargado de la página oficial de MySQL, http://dev.mysql.com/downloads/

El paquete descargado debe ser descomprimido y para que el la aplicación pueda ser instalada deben existir en el sistema el usuario y el grupo *mysql*, que se crean como se muestra a continuación:

```
# groupadd mysql
# useradd -g mysql mysql
```

Se configuran las variables de ambiente CFLAGS, CXX y CXXFLAGS

```
# CFLAGS="-O3 -pg"
# CXX=gcc
# CXXFLAGS="-O3 -felide-constructors -fno-exceptions -fno-rtti -pg"
# export CFLAGS CXX CXXFLAGS
```

Debe verificarse que las librerias de gcc, g++ y libncurses-dev, esten instaladas en el sistema, de no ser así, deben instalarse.

Enseguida se ejecuta configure desde el directorio de instalación de MySQL, la opción --with-ndbcluster debe ser incluida para que se active el manejo del motor de almacenamiento NDB.

```
#./configure --prefix=/usr/local/mysql51 --enable-assembler --with-mysqld-ldflags=-all-
static --with-ndbcluster
```

Ejecutar los comandos:

```
# make
# make install
```

Se copia el archivo de configuración al directorio /etc, en el ejemplo se indica la copia del archivo que configura un modelo de memoria mediano.

```
# cp support-files/my-medium.cnf /etc/my.cnf
```

Enseguida, desde el directorio a donde se instaló MySQL se realizan algunos cambios de permisos

```
# cd /usr/local/mysql51

# chown -R mysql .
# chgrp -R mysql .
```

Se inicializan las bases de datos:

```
# bin/mysql_install_db --user=mysql
Installing MySQL system tables...
OK
Filling help tables...
OK

To start mysqld at boot time you have to copy
support-files/mysql.server to the right place for your system

PLEASE REMEMBER TO SET A PASSWORD FOR THE MySQL root USER !
To do so, start the server, then issue the following commands:
/usr/local/mysql/bin/mysqladmin -u root password 'new-password'
/usr/local/mysql/bin/mysqladmin -u root -h lhopital password 'new-password'
See the manual for more instructions.
```

```
You can start the MySQL daemon with:
cd /usr/local/mysql ; /usr/local/mysql/bin/mysqld_safe &

You can test the MySQL daemon with mysql-test-run.pl
cd mysql-test ; perl mysql-test-run.pl

Please report any problems with the /usr/local/mysql/bin/mysqlbug script!

The latest information about MySQL is available on the web at
http://www.mysql.com
Support MySQL by buying support/licenses at http://shop.mysql.com
```

Se cambian algunos permisos:

```
# chown -R root .
# chown -R mysql var
```

Finalmente, para cada nodo MySQL y de datos deben existir el commando de activación de trabajo en cluster y la ubicación del nodo administrador del cluster.

```
# The MySQL server
[mysqld]

ndbcluster
ndb-connectstring=192.168.10.10

[MYSQL_CLUSTER]
ndb-connectstring=192.168.10.10
```

Glosario

ACID: En bases de datos se denomina ACID a la propiedad de una base de datos para realizar transacciones seguras. Así pues ACID compliant define a un sistema de gestión de bases de datos que puede realizar transacciones seguras. ACID es un acrónimo de *Atomicity*, *Consistency*, *Isolation* and *Durability*: Atomicidad, Consistencia, Aislamiento y Durabilidad en español.

Afinidad del procesador: Establece los procesadores en los que puede ejecutarse el subproceso asociado. En servidores con varias CPU se pueden configurar los grupos de aplicaciones para establecer la afinidad entre los procesos de trabajo y varias CPU y así utilizar de manera más eficiente las cachés de las CPU. La afinidad de procesador se utiliza junto con la configuración de máscara de afinidad de procesador para especificar las CPU.

Aislamiento: La propiedad de la ejecución de una transacción la cual determina que los efectos de una transacción sobre la base de datos son aislados de otras transacciones hasta que la primera complete su ejecución.

AIX: AIX (Advanced Interactive eXecutive) es un sistema operativo UNIX System V propietario de IBM. Inicialmente significaba "Advanced IBM Unix" pero probablemente el nombre no fue aprobado por el departamento legal y fue cambiado a "Advanced Interactive eXecutive". AIX es ejecutado en los servidores IBM eServers y pSeries, utilizando procesadores de la familia IBM POWER de 32 y 64bits. Algunas de las características únicas de AIX incluyen el Object Data Manager (ODM, una base de datos de información del sistema). La integración de AIX del "Logical Volume Management" (administrador de volumen lógico) dentro del núcleo está siendo incluido gradualmente a varios sistemas operativos libres similares a UNIX.

Atomicidad: La propiedad de procesamiento de transacciones donde o bien todas las operaciones de una transacción son ejecutadas o bien ninguna de ellas (todo o nada).

Callback: En programación de computadoras, un callback es un código ejecutable que es pasado como un argumento a otro código. Lo que permite que una capa de software de bajo nivel llame una subrutina (o función) definida en una capa de más alto nivel.

Commit: En el contexto de la ciencia de la computación y la gestión de datos, commit (acción de cometer) se refiere a la idea de hacer que un conjunto de cambios "tentativos, o no permanentes" se conviertan en permanentes. Un uso popular es al final de una transacción de base de datos.

Consistencia: es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto se ejecutan aquellas operaciones que no van a romper la reglas y directrices de integridad de la base de datos.

Contención: Competencia por recursos. El término es usado especialmente en redes para describir la situación donde dos o más nodos intentan transmitir un mensaje a través del mismo cable al mismo tiempo.
Un tipo de protocolo de red permite a los nodos competir por el acceso a red. Esto es, dos o más nodos podrían tratar de enviar mensajes a través de la red simultáneamente. El protocolo de contención define que pasa cuando esto ocurre. El protocolo de contención más usado es CSMA/CD usado por Ethernet.

Durabilidad: es la propiedad que asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer aunque falle el sistema.

Ext2: (*second extended filesystem* o "segundo sistema de archivos extendido") es un sistema de archivos para el kernel de Linux. El sistema de ficheros tiene un tipo de tabla FAT de tamaño fijo,

donde se almacenan los i-nodos. Los i-nodos son una versión muy mejorada de FAT, donde un puntero i-nodo almacena información del archivo (ruta o path, tamaño, ubicación física). En cuanto a la ubicación, es una referencia a un sector del disco donde están todos y cada una de las referencias a los bloques del archivo fragmentado. Estos bloques son de tamaño determinable cuando se crea el sistema de archivos, desde los 512 bytes hasta los 4 kB, lo cual asegura un buen aprovechamiento del espacio libre con archivos pequeños.

Failover: Es la capacidad de cambiar automáticamente a un servidor, sistema o red redundante o de reserva, ante la falla o terminación anormal de servidor, sistema o red activo. El failover ocurre sin la intervención humana y generalmente sin advertencia o alteración durante el cambio

Indirección: Es una técnica de programación. El concepto se basa en hacer referencia indirecta a los datos usando las direcciones de memoria que los contienen o mediante punteros que señalan hacia esos datos o a las direcciones que los contienen.

Inode: En computación, un inode es una estructura de datos sobre un sistema de archivos al estilo de Unix como el UFS. Un inode almacena información básica acerca de un archivo regular, directorio u otros objetos del sistema de archivos. Cuando un sistema de archivos es creado, estructuras de datos que contienen información acerca de los archivos son creadas. Cada archivo tiene un *inode* y es identificado por un número de inode (frecuentemente referenciado como "i-número" o "*inode*") en el sistema de archivos donde reside.

Paralelismo segmentado: Es cuando múltiples pasos dependen uno de otro, pero la ejecución puede ser traslapada y la salida de un paso es enviada como entrada del siguiente paso. La segmentación puede ser extendida para incluir cualquier número de pasos y puede ser extendida entre diferentes máquinas físicas.

Serializabilidad: El criterio de control de concurrencia el cual requiere que la ejecución concurrente de un conjunto de transacciones debe ser equivalente al resultado de la ejecución serial de esas transacciones.

Time-out: Una señal de interrupción generada por un programa o un dispositivo ha ha esperado un cierto tiempo por alguna entrada, pero no la ha recibido. Muchos programas ejecutan time-outs para no permanecer inactivos esperando por una entrada que nunca vendrá.

Referencias

[Abdelguerfi y Wong, 1998]

Abdelguerfi M., Wong K., Parallel Database Techniques, IEEE Computer Society Press, Los Alamitos, CA, 1998.

[APPLESHARE, 2007]

Mac OS 8 and 9 Developer Documentation: AppleShare, http://developer.apple.com/documentation/macos8/NetworkCommSvcs/AppleShare/appleshare.html

[Baer, 2005]

Baer H., Lumpkin G., “Parallel Execution in Oracle Database 10g Release 2”, An Oracle White Paper, Junio, 2005.

[Bancilhon *et al.*, 1987]

Bancilhon F., Briggs T., Khoshaflan S., Valduriez P., “FAD, a Powerful and Simple Database Language”, Proceedings of the 13th VLDB Conference, Brighton 1987.

[Baru *et al.*, 1995]

Baru C. K., Fecteau G., Goyal A., Hsiao H., Jhingran A., Padmanabhan S., Copeland G. P., Wilson W. G.,”DB2 Parallel Edition”, IBM Systems Journal, Vol. 34, No. 2, 1995, págs. 292-322.

[Banerjee *et al.*, 1979]

Banerjee J., Hsiao D. K., Kannan K., “DBC-A Database Computer for Very Large Databases”, IEEE Transactions on Computers, Vol. 28, No. 6, Junio, 1979, págs. 414-429.

[Blelloch y Maggs, 1996]

Blelloch G. E., Maggs B. M., “Parallel Algortihms”, Communications of the ACM, Vol. 28, No. 1, 1996, págs. 51-54.

[Boral *et al.*, 1990]

Boral H., Alexander W., Clay L., Copeland G., Danforth S., Franklin M., Hart B., Smit M., Valduriez P., “Prototyping Bubba, A Highly Parallel Database System”, IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 1, Marzo, 1990, págs. 4-24.

[Boutros y Desai, 1996]

Boutros B.S., Desai B.C., "A two-phase commit protocol and its performance", 7th International Workshop on Database and Expert Systems Applications (DEXA'96), 1996, págs 100-105.

[Braam, 1998]

Braam P. J., “The Coda Distributed File System”, Linux Journal, no. 50, 1998.

[Braam *et al*., 2000]
Braam P., Baron R., Harkes J., Schnieder M., The Coda HOWTO, ver. 1.01, http://www.coda.cs.cmu.edu/doc/html/coda-howto.html, Enero, 2000

[Ceri *et al*., 1987]

Ceri S., Pernici B., Wiederhold G., "Distributed Database Design Methodologies", Proceedings of the IEEE, Vol. 75, No. 5, Mayo, 1987, págs. 533-546.

[Codd, 1990]

Codd E. F., "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM, Vol 13, No. 6, Junio 1990, págs. 377-387.

[Cormen *et al.,* 2001]

Cormen T. H., Leiserson C. E., Rivest R. L., Stein C., Introduction to Algorithms, The MIT Press Cambridge, Massachusetts, Second Edition, McGraw-Hill Book Company, 2001.

[Dailey, 2004]

Dailey L., "Open Source Databases Move into the Marketplace", IEEE Computer Society Press, Los Alamitos, CA, USA, Vol. 37, No. 7, págs. 13-15.

[Davies y Fisk, 2006]

Davies A., Fisk H., MySQL Clustering, Indianapolis,Indiana, USA: MySQL Press, 2006.

[De Witt, 1979]

De Witt D. J., "Direct - A Multiprocessor Organization for Supporting Relational Database Management Systems", IEEE Transactions on Computers, Vol. 28, No. 6, Junio, 1979, págs 395-406.

[De Witt *et al.*, 1990]

De Witt D. J., Ghandeharizadeh S., Schneider D., Bricker A., Hsiao H., Rasmussen R., "The Gamma Database Machine Project",IEEE Knowledge and Data Engineering, Vol. 2, No. 1, Marzo, 1990.

[De Witt y Gray, 1992]

De Witt David J., Gray Jim. "Parallel Databases Systems: The Future of High Performance Database Processing", Communications of ACM, Vol. 6, Junio 1992, págs. 85-98.

[Doeppner, 1996]

Doeppner T. W. Jr., "Distributed File Systems and Distributed Memory", ACM Computing Surveys, Vol. 28, No. 1, Marzo, 1996, págs. 229-231.

[Elhardt y Bayer, 1984]

Elhardt K., Bayer R., "A Database Cache for High Performance and Fast Restart in Database Systems", ACM Transactions on Database Systems, Vol. 9, No. 4, Diciembre, 1984.

[Forum Nokia, 2007]

Forum Nokia Pro Success Story, “Oracle ports Berkeley DB to S60 Using Open C”, 2007.

[Frank , 1999]

Frank L., “Atomicity Implementation in Mobile Computing”, Tenth International Workshop on Database and Expert Systems Applications, Septiembre, 1999, págs. 105-113.

[Ghandeharizadeh, 1990]

Ghandeharizadeh S., De Witt D. J., “Hybrid-Range Partitioning Strategy: A New Declustering Strategies”, Proceedings of the Sixteenth International Conference on Very Large Data Bases”, Melbourne, Australia, Agosto, 1990.

[Graefe y Shapiro, 1991]

Graefe G., Shapiro L.D., “Data Compression and Database Performance”, [Proceedings of the 1991] Symposium on Applied Computing, 1991, Abril, 1991, págs. 22-27.

[Gray, 1989]

Gray J., “Transparency in its Place – The Case Against Transparent Access to Geographically Distributed Data. Technical Report TR89.1, Cupertino, Calif.: Tandem Computers Inc., 1989.

[Gropp y Lusk, 1995]

Gropp W. D., Lusk E., “Experiences with the IBM SP1”, IBM Systems Journal, Vol. 34, No. 2, 1995.

[Howard *et al.*, 1988]

Howard J.H., Kazar M.L., Menees S.G., Nichols D.A., Satyanarayanan M., Sidebotham R.N., West M.J., ”Scale and Performance in a Distributed File System”, ACM Transactions on Computer Systems, Vol. 6, No. 1, Febrero 1988, pp. 51-81.

[Jiménez-Peris *et al.*, 2001]

Jiménez-Peris R., Patino-Martínez M., Alonso G., Kemme B., “How to Select a Replication Protocol According to Scalability, Availability and Communication Overhead”, Proceedings. 20th IEEE Symposium on Reliable Distributed Systems, 2001, págs. 24-33.

[Levy y Silberschartz, 1990]

Levy E., Silberschartz A., “Distributed File Systems: Concepts and Examples”, ACM Computing Surveys, Vol. 22, No. 4, 1990, págs 321-374.

[LOTUS, 2008]

IBM, Lotus Notes; http://www-306.ibm.com/software/lotus/category/email/

[Mattson *et al.,* 2005]

Mattson T. G., Sanders B. A., Massingill B. L., Patterns for Parallel Programming, Software Patterns Series, Pearson Education Inc, 2005

[Mohan *et al.*, 1986]

Mohan C., Lindsay B., Obermarck R., “Transaction Management in the R* Distributed Database Management System”, ACM Transactions on Database Systems, Vol. 11, No. 4, Diciembre 1986, págs. 378-396.

[MYSQL, 2008]

MySQL Home Page, http://dev.mysql.com/

[MySQL 5.1, 2007]

MySQL 5.1 Reference Manual, http://dev.mysql.com/doc/refman/5.1/en/index.html

[MySQL Press Releases, 2003]

MySQL Press Releases, “Westone Amplifies Database Performance with MySQL”, Enero, 2003.

[NCUBE]
nCube Corporation, http://www.npac.syr.edu/nse/hpccsurvey/orgs/ncube/ncube.html

[NETWARE, 2008]

Novell: Worldwide; http://www.novell.com

[ORACLE, 2008]

Oracle's History: Innovation, Leadership, Results;
http://www.oracle.com/corporate/story.html

[Özsu y Valduriez, 1991]

Özsu M. Tamer, Valduriez Patrick, “Distributed Database Systems: Where Are We Now?”, IEEE Computer, Vol. 24, No. 8, 1991.

[Özsu y Valduriez, 1996]

Özsu M. Tamer, Valduriez Patrick, “Distributed and Parallel Database Systems”, ACM Computing Surveys, Vol. 28, No. 1, págs. 125-128, 1996.

[Özsu y Valduriez, 1999]

Özsu M. Tamer, Valduriez Patrick, Principles of Distributed Database Systems, Second Edition, Prentice Hall, Inc., Upper Saddle River, New Jersey, USA.

[Pinkerton *et al*., 1990]

Pinkerton C. B., Lazowska E. D., Notkin D., Zahorjan J., “A heterogeneous distributed file system”, Conference on Distributed Computing Systems, 1990. Proceedings., 10th International, Junio, 1990, págs. 424 – 431.

[Ramos-Paz, 2207]

Ramos-Paz A., “Técnica para la Generación Automática de Ecuaciones Diferenciales No Autónomas para Representar el Comportamiento Dinámico de Sistemas Eléctricos No-Lineales Incorporando Herramientas Avanzadas de Cómputo”, Tesis Doctoral, Enero, 2007.

[Ronström, 2005]

Ronström M., “High availability features of MySQL Cluster”, Senior Software Architect, White Paper MySQL AB.

[Sandberg,1986]

Sandberg R., “The Sun Network Filesystem: Design, Implementation and Experience”, Sun Microsystems, Inc, 1986

[Satyanarayanan, 1989]

Satyanarayanan M., “A Survey of Distributed File Systems”, Annual Review of Computer Science. Annual Reviews, Inc, 1989. También disponible como Tech. Rep. CMU-CS-89-116, Carnegie Mellon University School of Computer Science, February, 1989. http://citeseer.ist.psu.edu/satyanarayanan89survey.html

[Satyanarayanan *et al.*, 1990]

Satyanarayanan M., Kistler J. J., Kumar P., Okasaki M. E., Siegel E. H., Steere D. C., “Coda: A Highly Available File System for a Distributed Workstation Environment”, IEEE Transactions on Computers, Vol. 39, No. 4, Abril, 1990, págs. 447-459.

[Satyanarayanan, 1990b]

Satyanarayanan M., “Scalable, Secure, and Highly Available Distributed File Access”, IEEE Computer, Vol. 23, No. 5, Mayo 1990, págs 9-21.

[Satyanarayanan, 1992]

Satyanarayanan M., “The Influence of Scale on Distributed File System Design”, IEEE Transactions on Software Engineering, Vol. 18, No. 1, Enero 1992,

[Schumacher, 2006]

Schumacher R., “Improving Database Performance with Partitioning”, Developer Articles, http://dev.mysql.com/tech-resources/articles/

[Skeen, 1981]

Skeen D., “Nonblocking Commit Protocols”, SIGMOD '81: Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data, 1981, págs. 133-142.

[Stonebraker, 1986]

Stonebraker M., “The Case for Shared Nothing”, Database Engineering, Vol 9, No. 1, 1986.

[Stonebraker, 1989]

Stonebraker M., “Future Trends in Database Systems”, IEEE Trans. Knowledge and Data Eng., Marzo 1989, Vol. 1, No. 1, págs 33-44.

[Thakkar y Sweiger, 1990]

Thakkar, S.S. Sweiger, M., “Performance of an OLTP application on Symmetry multiprocessorsystem”, Proceedings. 17th Annual International Symposium on Computer Architecture, 1990 Mayo, 1990, págs: 228-238

[Tamura *et al.* 1996]

Tamura T., Nakamura M., Kitsuregawa M., y Ogawa Y., “Implementation and performance evaluation of the parallel relational database server SDC-II”. In Proceedings of International Conference on Parallel Processing, 25th, Vol. 1, págs. 212-221, 1996.

[Tandem Database Group, 1987]

Tandem Database Group, “NonStop SQL, a Distributed, High-Performance, High-Availability Implementation of SQL, Technical Report 87.4, Abril 1987.

[TERADATA]

Teradata®, http://www.teradata.com

[Többicke, 1994]

Többicke R., “Distributed File Systems: Focus on Andrew File System/Distributed File Service (AFS/DFS)”, Thirteenth IEEE Symposium on Mass Storage Systems, Junio, 1994, págs. 23-26.

[UBUNTU, 2008]

Ubuntu Home Page, http://www.ubuntu.com/

[Witkowski *et al.,* 1993]

Witkowski A., Cariño F., Kostamaa P.,”NCR 3700 – The Next-generation Industrial Database Computer”,Proceedings of the 19th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc, San Francisco, CA., USA, 1993, págs. 230-243.