Universidad Central "Marta Abreu" de Las Villas
Facultad de Matemática – Física – Computación
Departamento de Ciencia de la Computación

Uso de Sistemas de Gestión de Bases de Datos Paralelas para la Visualización Científica.

Tesis presentada en opción al grado científico de
Master en Ciencia de la Computación

Autor: Lic. Saumel Enriquez Caro

Tutor: Dr. Carlos Pérez Risquet

Santa Clara
2008

Resumen

En los últimos años, los avances en la tecnología de la información han facilitado la obtención de grandes cantidades de información. La manipulación, el almacenamiento, el acceso eficiente y el análisis de esos datos representan una tarea que constituye un gran reto. La visualización científica ha surgido como una herramienta capaz de procesar y analizar estos grandes volúmenes de datos hasta el orden de los terabytes.

Tradicionalmente se han empleado formatos de archivos binarios secuenciales para gestionar los datos utilizados en la visualización. Sin embargo el trabajo con estos modelos no resulta suficiente ante las demandas de la comunidad científica actual. Por lo que se necesita una solución integral que combine las ventajas de los Sistemas Gestores de Bases de Datos (SGBD), la eficiencia de los formatos nativos y las potencialidades del procesamiento paralelo.

En este trabajo se hace un análisis de la factibilidad del uso de los SGBD Paralelos de código abierto para el almacenamiento y visualización de los datos científicos. Posteriormente se seleccionan aquellos sistemas con herramientas que resultan más eficientes para su implementación en un cluster de computadoras y se realiza la implementación de un caso de estudio.

Abstract

In the last years, the advances in the technology of the information have facilitated the obtaining of big quantities of information. The manipulation, the storage, the efficient access and the analysis of those data represent a task that constitutes a great challenge. The Scientific Visualization provides methods and tools able to process and to analyze that huge amount of data even in the order of the terabytes.

Traditionally flat binary sequential files have been used to handle the data used on the visualization. However the work with these models is not enough before the demands of the current scientific community. For what an integral solution is needed that combines the advantages of the Databases Management System (DBMS), the efficiency of the native formats and the potentialities of the parallel processing.

In this work an analysis of the feasibility of the use of Parallel Object-Relational DBMS of open source is made for the storage and visualization of the scientific data. Then some systems are selected with tools that are more efficient for their implementation in a cluster of computers and it's implemented a case of study with this tools.

Tabla de Contenidos

Lista de Figuras

Introducción

Los volúmenes de datos generados por las aplicaciones científicas crecen de manera exponencial cada año. Los avances alcanzados en instrumentos, detectores y tecnología de computadoras están creando repositorios de datos del orden de los terabytes en muchas disciplinas. La visualización científica emerge como una alternativa para extraer información de estos grandes volúmenes de datos, basada en la generación, a partir de ellos, de imágenes, ya sean estáticas o en movimiento. Para que este proceso sea efectivo se necesitan formas eficientes de almacenamiento y gestión de los datos.

Una alternativa común usada en los primeros algoritmos de cálculos científicos fue el empleo de archivos planos secuenciales para el almacenamiento de los datos de entrada y salida. Esta solución, aunque simplifica el acceso a la información, es ineficiente cuando se almacenan y procesan grandes conjuntos de datos al presentar problemas de interoperabilidad.

La necesidad de leer y escribir datos científicos de forma eficiente e independiente de la arquitectura condujo a la creación de nuevos modelos. Estos modelos incluyen estructuras de datos que soportan grandes conjuntos, arreglos multidimensionales, gran variedad de tipos de datos multidimensionales, multivariados, multievaluados y métodos para incorporar metadatos. Como ejemplos de estos estándares existen NetCDF, HDF y FITS (Bose and Frew, 2005). Cada uno incluye bibliotecas que encapsulan los archivos correspondientes y brindan una forma independiente de la plataforma, para crear, acceder, buscar y visualizar los datos. Estas bibliotecas están disponibles mediante una interfaz de programación de aplicaciones que puede ser usada desde varios lenguajes, siendo los más comunes C, C++ y Java.

Los estándares para el almacenamiento facilitan el intercambio de datos entre los científicos. Sin embargo, es común que cada disciplina o comunidad científica se incline por un formato específico, así como las herramientas construidas para él, lo que hace que el intercambio fuera del grupo no sea basado en estándares que favorezcan su manipulación (Gray, 2002).

Adicionalmente, aunque las ventajas sobre el uso de archivos planos son considerables, este modelo aún presenta el problema de la generación de un gran número de archivos, debido a que los científicos acostumbran a generar un archivo por cada simulación, provocando que cuando el sitio de almacenamiento no coincide con el de procesamiento, deban moverse grandes cantidades de archivos de un lugar a otro. Los archivos se organizan en directorios, por lo que el sistema de archivos no contiene más metadatos que la estructura jerárquica de nombres. Esto promueve un modelo de datos que no se beneficia del surgimiento de las nuevas herramientas de análisis. En este modelo el análisis de datos se realiza buscando primero todos los archivos relevantes, abriendo cada uno de ellos, buscando los datos necesarios y moviéndose hacia el próximo.

Cuando todos los datos están en memoria, o en un archivo intermedio, el programa puede comenzar el análisis. El proceso de "filtra, después analiza", hace que el análisis de grandes volúmenes de datos con herramientas procedurales convencionales sea más lento a medida que el volumen de los datos aumenta. Esta forma de acceso impide las búsquedas temporales, espaciales, asociativas y paralelas, así como el uso de un lenguaje de consulta de alto nivel (Gray and Liu, 2005).

Varios de los problemas mencionados anteriormente pueden ser resueltos utilizando SGBD. Estos sistemas están diseñados para minimizar el tiempo de las búsquedas, hacer balance de carga y manejar grandes volúmenes de datos (Hellman, 2007). Además, permiten el acceso simultáneo a los datos desde varias aplicaciones, brindando un ambiente adecuado para compartir los mismos. Los SGBD también soportan poderosos lenguajes de consultas no procedurales así como diferentes métodos de almacenar consultas de forma pre-compiladas (procedimientos almacenados, funciones y vistas que adicionalmente son optimizadas en los algoritmos más modernos), independencia física y lógica de los datos, y variadas herramientas para el diseño y manipulación. Sin embargo, la comunidad científica no ha sido proclive a usar los SGBD comerciales. Entre las razones de mayor peso se encuentran que los SGBD Relacionales no brindan soporte para arreglos multidimensionales y su rendimiento al trabajar

con grandes arreglos es bajo. Sin embargo, el surgimiento de los SGBD Objeto Relacionales ha abierto las puertas a la inclusión de conjuntos de datos científicos dentro de los SGBD. Por otra parte, las aplicaciones científicas en general y las de visualización en particular, difieren de las aplicaciones comerciales en la forma de procesar los datos, pues en estas últimas la información cambia constantemente mientras que en las aplicaciones científicas los datos se almacenan de forma histórica y generalmente no cambian, lo que hace que la capacidad de almacenamiento y el poder de procesamiento que se necesite sea muy alto. Para satisfacer esta demanda, se han empleado diversas tecnologías que coinciden fundamentalmente en utilizar arquitecturas paralelas, aumentando considerablemente los índices de rendimiento y disponibilidad de estos sistemas, de forma especial, en la visualización científica se han utilizado para paralelizar el proceso de rendereo, que es una de las tareas que mayor demanda de procesamiento presenta (Allcock, 2001).

Como se ha observado anteriormente, se han utilizado varios enfoques para el manejo de los datos científicos, entre los que se destacan el uso de archivos binarios, pero no se ha definido un modelo que sea capaz de realizar esta tarea de forma totalmente eficiente. Es por eso que en esta investigación se plantea el siguiente problema de investigación:

El proceso de visualización científica es una moderna y poderosa herramienta para el análisis de grandes volúmenes de datos, que requiere de una eficiente gestión de la información. Tradicionalmente se han utilizado diversas tecnologías y herramientas por la comunidad científica que no satisfacen los requerimientos que presenta la visualización en la actualidad, principalmente por la incompatibilidad entre las diferentes versiones, la falta de información acerca de los datos que almacenan y la reducida capacidad que presentan estas alternativas para manejar la información histórica almacenada. No existe hasta el momento una solución integral que combine al mismo tiempo las ventajas de los SGBD (optimización automática de consultas, ejecución de consultas en paralelo, uso de lenguajes de consulta no procedurales, uso de índices, entre otras), la eficiencia que brindan los formatos nativos de almacenamiento de datos empleados en la visualización científica y las potencialidades del procesamiento paralelo.

Este problema científico conduce a las siguientes preguntas de investigación:

1. ¿Cómo incorporar datos científicos en un SGBD, sin perder la eficiencia que ofrecen los formatos nativos de almacenamiento de datos empleados en la visualización científica?
2. ¿Cuáles SGBD son en principio apropiados para incorporarles datos científicos como uno de sus tipos primitivos de datos?
3. ¿Cómo implementar la estructura de los datos científicos en una base de datos paralela?
4. ¿Cuán eficiente resulta incorporar datos científicos en una base de datos paralela con respecto a las soluciones existentes en la actualidad, que emplean formatos nativos para el almacenamiento de la información?

A partir de estas interrogantes se pueden plantear los objetivos de esta investigación.

Objetivo General

Determinar la factibilidad del uso de un sistema de bases de datos paralelo para gestionar datos en el área de la visualización científica e implementar un caso de estudio en un clúster de computadoras.

Objetivos Específicos

1. Seleccionar, entre las tecnologías empleadas para el almacenamiento y gestión de la información para la visualización científica, las más apropiadas para su integración en un SGBD.
2. Determinar las herramientas de software libre disponibles para implementar un sistema de bases de datos paralelos que gestione datos científicos.
3. Determinar la manera más apropiada de vincular los formatos nativos de datos científicos con un SGBD.

4. Desarrollar un caso de estudio en el que se realicen consultas de visualización científica en una base de datos implementada sobre un clúster de computadoras.
5. Implementar una herramienta que permita importar datos almacenados en el formato HDF a un SGBD paralelo.

Capítulo 1 Almacenamiento y Gestión de Datos Científicos

1.1 Introducción

Como se ha mencionado, los avances de la tecnología en el área de la computación y la informática han hecho imposible analizar toda la información que diariamente es recopilada. El volumen de los datos generados por instrumentos científicos y en simulaciones ha alcanzado el orden de los terabytes, ocasionando dificultades para procesar esta cantidad de información (Gallop, 1994). En general se ha demostrado que de toda la información generada, sólo una cuarta parte se almacena, y de ésta a su vez sólo una cuarta parte realmente se analiza. Evidentemente el por ciento de información que se pierde es significativo.

Lo anterior ha provocado que se realicen investigaciones con el objetivo de encontrar nuevos métodos que permitan aumentar la capacidad de procesamiento de información. La visualización científica emerge como una representación novedosa, que permite detallar dinámicamente las principales características de los datos, así como experimentar con lo no observable para transmitir ideas tanto abstractas como concretas, permitiendo así enseñar, organizar y simular objetos y situaciones que serían imposible de lograr utilizando métodos estadísticos (Yurcik, 2007). Como la cantidad de información que se necesita visualizar es considerable en la mayoría de los casos, es de vital importancia que el almacenamiento se realice de forma eficiente. Tradicionalmente para gestionar los datos utilizados en la visualización científica se han destacado dos vertientes, los formatos de datos específicos y las bases de datos científicas.

En este capítulo se profundiza en el proceso de visualización científica, en las bases de datos que utilizan la visualización como herramienta para el procesamiento y análisis de la información, así como la forma en que implementan las tecnologías para gestionar los datos que utilizan.

1.2 Visualización de grandes volúmenes de datos

La visualización es el proceso de generar una imagen, ya sea mental o real, de algo abstracto o invisible. Cuando la imagen generada es real, la cantidad de información que se transmite es mucho mayor que si se utilizan otros métodos, pues la potencialidad del sistema visual de las personas es ilimitado.

Existen varios tipos de visualización pero se profundizará en los aspectos fundamentales de la visualización científica, siendo esta un área de gran importancia en la computación. Al igual que en otras áreas, gracias al avance del software y a los bajos costos del *hardware*, el desarrollo del análisis visual de datos ha sido considerable.

El objetivo principal de la visualización científica es lograr simplificar el análisis, comprensión y comunicación de los datos, modelos y conceptos abstractos en el área de la ciencia y la ingeniería, suministrando así a los científicos, representaciones visuales apropiadas para mostrar correlaciones internas de los datos, las cuales no serían detectadas si no pudieran visualizarse (Gallop, 1994).

Las herramientas y técnicas de visualización han sido utilizadas de forma muy efectiva y coherente para analizar grandes volúmenes de datos, en muchas ocasiones multidimensionales, de forma que permitan al usuario extraer resultados significativos de forma rápida y fácil, a través de la transformación de datos científicos y abstractos en imágenes. Las técnicas de visualización evitan la omisión de datos importantes diferenciándolos de los inservibles, facilitan la compresión de conceptos complicados y la comunicación entre científicos.

En el área de la visualización científica se han desarrollado potentes herramientas clasificadas de acuerdo a su propio campo de aplicación definido por el tipo de datos que procesan principalmente volúmenes o fluidos.

1.2.1 Visualización de Volúmenes

La visualización de volúmenes se refiere generalmente a la visualización de campos escalares. Se extiende desde el examen de datos científicos a la reconstrucción de datos dispersos y a la representación de objetos geométricos sin una descripción matemática de su superficie, permitiendo el análisis del interior de un volumen utilizando técnicas como las de cortes y transparencia. Una cuestión no tan madura en la visualización de volúmenes la constituye el problema de la dimensionalidad. Cuando la dimensión de los datos excede las tres dimensiones, la visualización no puede manejarse usando los métodos tradicionales y se requieren nuevas técnicas que permitan simular imágenes de acuerdo a las necesidades de la aplicación.

1.2.2 Visualización de Fluidos

Esta técnica se utiliza para la visualización en general de sistemas dinámicos, es decir, aquellos sistemas en los que hay involucrados datos que evolucionan en el tiempo. El comportamiento cualitativo de dichos sistemas puede comprenderse adecuadamente a partir de la estructura de la evolución temporal de sus trayectorias. Este tipo de visualización científica es muy eficiente pues abarca frecuentemente una gran cantidad de datos que resultan muy difíciles de comprender. Debido a que la variación en el tiempo muchas veces es invisible, las técnicas de visualización de fluidos están determinadas por la interacción del flujo con la luz que puede ser expresado de dos maneras:

1) La luz puede ser esparcida por las moléculas del fluido o por partículas trazadoras depositadas en él.
2) La luz cambia sus propiedades debido al comportamiento óptico del flujo, de tal manera que la luz transmitida a través de él es diferente con respecto a la luz incidente.

Las técnicas de Visualización de Fluidos usadas en la ciencia y la industria están basadas en tres principios básicos: el esparcimiento de la luz por las partículas trazadoras; cambios del índice de refracción producidos por el fluido; y la interacción del flujo con superficies sólidas. De forma general representan una potente herramienta dentro de la visualización científica que ha sido ampliamente utilizada por todas las ramas de la sociedad (Yurcik, 2007).

1.3 Visualización Distribuida

Como se ha mencionado el volumen de los datos utilizados en el área de la computación grafica, específicamente para la visualización científica es notable, por lo que se requieren tecnologías novedosas que permitan de forma transparente a los usuarios distribuir la información entre varios sistemas y paralelizar los complejos algoritmos de rendereo, es por esto que las recientes investigaciones centran su atención en la visualización distribuida.
Es importante en primer lugar definir qué es la visualización distribuida:

> *"Utilizar una o más computadoras para darle a un usuario una imagen o representación visual de un conjunto de datos"*(Heijmans, 2002).

Una visión general de los distintos métodos para implementar esta tecnología se puede observar en la figura1.1.

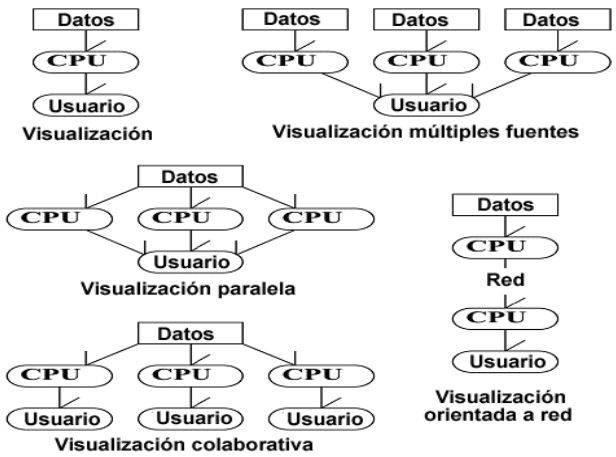

Figura 1.1 Visión general de los tipos de visualización distribuida

La visualización paralela a menudo requiere un alto costo computacional, definido principalmente por el tiempo necesario para realizar una tarea, este tiempo puede ser disminuido utilizando varios procesadores conectados a través de una red. La visualización colaborativa puede ser muy útil para facilitar el trabajo de un equipo de personas simultáneamente. Esta forma de trabajo elimina considerablemente la redundancia y aumenta la calidad de los resultados. La visualización orientada a la red permite al usuario crear una imagen de datos que residen en cualquier lugar fuera de su computadora. Con la visualización a partir de múltiples fuentes se pueden combinar resultados de varios conjuntos de datos en una misma imagen. Este trabajo centra su atención en la visualización a partir de múltiples fuentes y la visualización paralela.

La estructura de un sistema de visualización puede ser comparada fácilmente con la estructura general de un sistema de información. La base consta de datos y metadatos descriptivos. El usuario tiene una interfaz mediante la cual puede observar los metadatos, los datos, y controlar las rutinas de fondo que actúan sobre la información. En un sistema de este tipo, al motor de visualización se le llama rutina de fondo. Es importante dividir la interfaz en dos partes, la interfaz gráfica de usuario (GUI) y el observador.

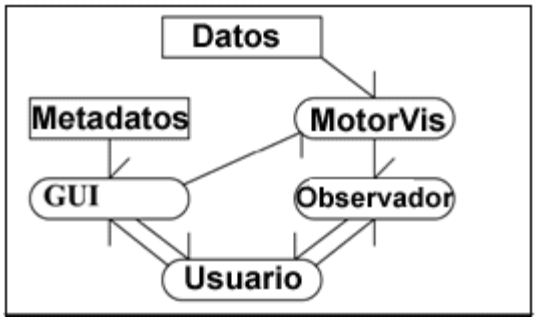

Figura 1.2 Control de flujo y visualización de datos

El usuario interactúa con la GUI para acceder a los metadatos y seleccionar los parámetros del motor de visualización. Los datos son transformados por el motor de visualización y mostrados a la vista del usuario por el observador.

En la figura 1.3 se muestra la estructura de un sistema de visualización distribuido ilustrando el flujo de metadatos a través de él. El servidor le envía los metadatos al cliente, y estos son mostrados al usuario por la GUI.

Los metadatos tienen gran importancia, porque permiten al usuario escoger correctamente los datos y determinar los parámetros que se deben tener en cuenta para la visualización. Sin embargo muchos sistemas omiten este proceso y no especifican qué metadatos están disponibles.

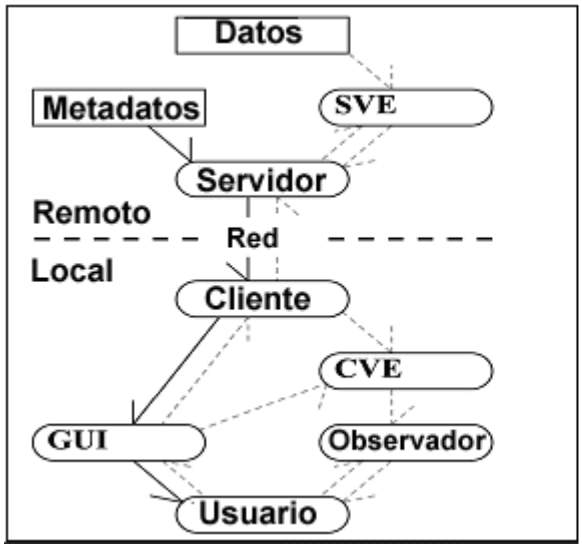

Figura 1.3 Flujo de metadatos

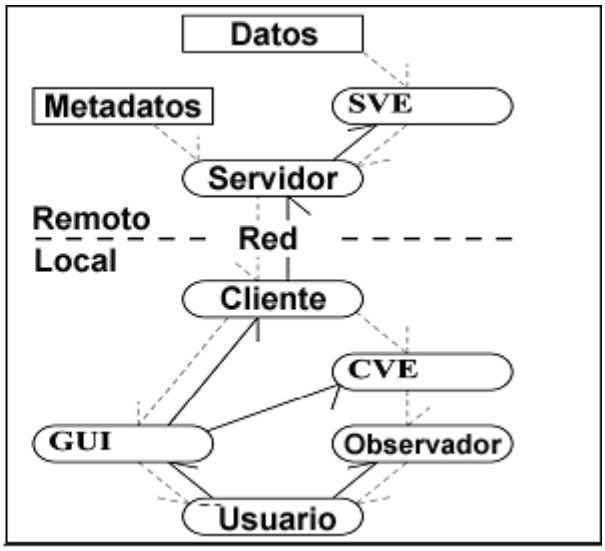

Figura 1.4 Control de flujo

En la figura 1.4 se describe el control de flujo de un sistema de visualización. El usuario controla al observador, al motor de visualización del lado del cliente (CVE) y al motor de visualización del lado del servidor (SVE) a través del GUI.

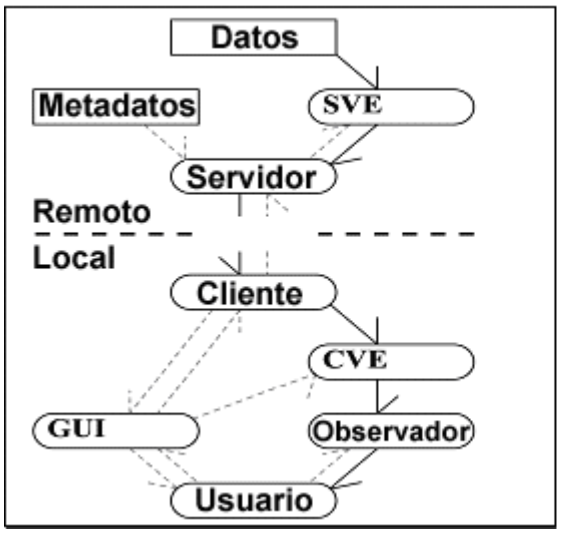

Figura 1.5 Flujo de datos

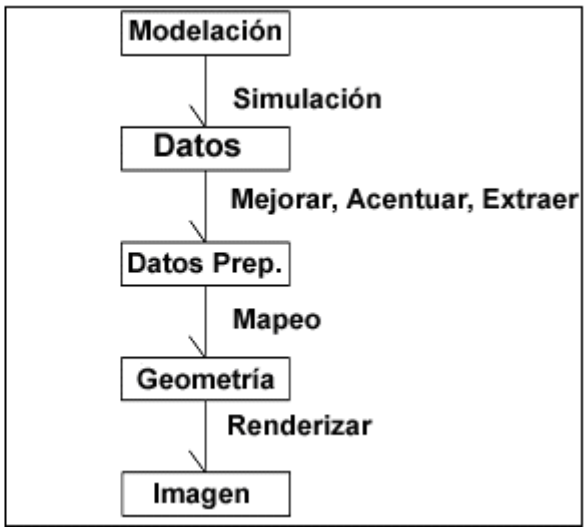

Figura 1.6 Tubería de Visualización

En la figura1.5 se muestra el flujo de datos a través del sistema. Los datos son procesados por el SVE, enviados del servidor al cliente, procesados por el CVE y exhibidos por el observador. De forma alternativa se muestra en la figura 1.6 el flujo de datos a través de la tubería de visualización, que transforma los datos de entrada en una imagen de dos o tres dimensiones, pasando por los motores de visualización del lado del cliente y del servidor, y por último del observador. El proceso de visualización puede efectuarse en cualquiera de los dos motores por lo que definir en cual de ellos se efectúa es un asunto importante. Entonces, se presentan dos criterios decisivos, definir la estructura de un sistema de visualización distribuido y el control que tiene el usuario sobre el proceso de visualización (Gong et al., 2004).

Cuando los datos están en un formato que el SVE no puede comprender directamente, es necesaria la transformación de los mismos para realizar la visualización, por lo cual se debe disponer de una herramienta de conversión o se debe usar un módulo especial para importar en el motor de visualización. Conceptualmente se puede colocar la conversión dentro de la fase de preparación. En ese caso no hay necesidad de hacer distinción con otros pasos de la visualización.

Existen tres niveles básicos de control:

- El usuario controla al observador: Éste es el nivel más bajo de interacción. El usuario puede controlar sólo su punto de vista, pero no tiene el control de la visualización. Por consiguiente la visualización puede llamarse apenas interactiva.

- El usuario controla la visualización en el lado del cliente, si se envían todos los datos al cliente, la interacción se realiza según las necesidades del usuario, él puede cambiar la tubería de visualización y sus parámetros.

- El usuario controla la visualización en el lado del servidor, no se envían los datos al cliente, sino una selección o incluso el resultado de una visualización, el usuario debe poder controlar la visualización del servidor mediante un formulario ofrecido para que seleccione los parámetros según su interés.

1.3.1 Tubería de Renderizado Paralelo

Un proceso paralelo típico de renderizar un volumen consta de cuatro pasos. En primer lugar se realiza la lectura de los datos en disco y la distribución hacia los procesadores. Cada procesador recibe un subconjunto del volumen de los datos. En el siguiente paso, cada procesador renderiza el subvolumen asignado en una imagen parcial que es totalmente independiente de los demás procesadores. Después, se desarrolla el proceso de combinación, que generalmente requiere comunicación entre los procesadores, donde se mezclan las imágenes para formar el resultado final. Por último se muestra la imagen al usuario o se almacena para su análisis posterior.

Dado un volumen genérico que debe ser renderizado en paralelo y un sistema de varios procesadores, hay tres formas posibles de administrar los procesadores para renderizar los volúmenes de datos, teniendo en cuenta el tiempo necesario para obtener la imagen final (Hansen and Johnson, 2005).

El primer método consiste simplemente en ejecutar el volumen a renderizar como una secuencia de subconjuntos uno tras otro. En cualquier punto de tiempo, todo el sistema está dedicado al renderizado de un volumen particular. El paralelismo sólo se asocia con el renderizado de un solo volumen de datos, en esencia solamente es explotado el paralelismo intravolumen.

El segundo método toma una estrategia opuesta, renderizando varios volúmenes de datos simultáneamente, cada uno en un procesador diferente. Esta alternativa sólo explota paralelismo intervolumen, y está limitado por el espacio principal de memoria de cada procesador.

Para lograr un renderizado óptimo, se deben balancear dos factores en el funcionamiento, el uso eficiente de los recursos y la paralelización; Esto sugiere explotar ambos tipos de paralelismo, el paralelismo intravolumen y el paralelismo intervolumen. En lugar de usar todos los procesadores para el renderizado colectivo de un volumen a la vez, se forma un proceso de renderizado de tubería dividiendo los procesadores en grupos de renderizado de múltiples volúmenes concurrentemente. De este modo, el tiempo global de renderizado puede ser minimizado, porque las tareas de la tubería de renderizado son superpuestas.

El tercer método es un híbrido, en el cual los nodos de todos los procesadores están subdivididos en una cantidad determinada de grupos, cada uno con un volumen de renderizado. La elección óptima de esta cantidad generalmente depende del tipo y la escala del sistema paralelo, así como del tamaño del conjunto de datos. La estrategia óptima de partición de discos para minimizar el tiempo global de renderizar puede ser caracterizada por el modelo de funcionamiento y revelada con un estudio experimental, lo cual demuestra que el tercer método ciertamente es el mejor de los tres.

1.3.2 Flujo de Datos

La teoría de Flujo de Datos (FD), es el método más usado para ejecutar procesos independientes de subconjuntos de datos de grandes dimensiones. Muchas veces es el único método factible frente a situaciones donde el tamaño de un conjunto de datos excede la capacidad de los recursos disponibles en cuanto a memoria y espacio de intercambio. Por ejemplo, en conjuntos de datos científicos, especialmente series de tiempo, que fácilmente exceden la cantidad de memoria disponible. La principal ventaja de este método se centra en que todos los conjuntos de datos de cualquier tamaño pueden ser tratados exitosamente. Sin embargo a menudo requiere largo tiempo de ejecución y no prevé la exploración interactiva de los datos. El uso de un subsistema de Entrada-Salida de alto rendimiento específico es ventajoso para mejorar el funcionamiento global de los algoritmos de flujo.

1.3.3 Paralelismo de Tareas

Como lo indica su nombre, con el paralelismo de tareas, los módulos independientes de una aplicación pueden ser ejecutados de forma paralela. Este proceso requiere un algoritmo que resuelva tareas independientes y múltiples recursos computacionales. La principal ventaja de esta técnica es que permite ejecutar tareas de visualización paralelamente desde múltiples fuentes de datos distribuidos. Sin embargo presenta el inconveniente de que el número de tareas que pueden ser tratadas tiene que ser igual al número de procesadores disponibles. El paralelismo de tarea es usado eficazmente en la edición de videos, donde varios marcos en una producción animada son renderizados de forma simultanea. Las elección específica del *hardware* para mejorar el funcionamiento del paralelismo de tareas depende de los detalles que se requieran.

1.3.4 Paralelismo de Tubería

El paralelismo de tubería ocurre cuando un número de módulos en una aplicación se ejecutan paralelamente pero en subconjuntos independientes de datos. Este método es más conveniente para situaciones donde hay múltiples tareas heterogéneas. Esta estrategia permite el uso paralelo de los recursos de cómputo. Por ejemplo, un proceso puede estar leyendo de disco, otro proceso computando resultados utilizando el procesador, y un tercer proceso usando una tarjeta aceleradora de gráficos. Es importante sin embargo señalar que puede ser difícil balancear el tiempo de ejecución requerido por las distintas etapas. En una tubería desnivelada, la etapa más lenta directamente afecta la función global. Además, la longitud de la tubería limita directamente al índice de paralelismo que puede ser logrado, y se deben tener tantos procesadores disponibles como etapas halla en la tubería. Para minimizar el tiempo de ejecución, hay que mover rápidamente los datos de una etapa de la tubería a la siguiente. Esta estrategia requiere el uso de arquitecturas de memoria compartida y una red de interconexión de alta velocidad entre los procesadores.

1.3.5 Paralelismo de datos

Mediante el paralelismo de datos, el código dentro de cada módulo de una aplicación se ejecuta simultáneamente. Este proceso requiere que el conjunto de datos sea subdividido en múltiples procesos que ejecutan el mismo algoritmo en las partes resultantes concurrentemente. El paralelismo de datos puede ser implementado como una extensión de la técnica de descomposición de datos descrita en la sección de flujo de datos (Chen and Liu, 2006). En este caso, los datos se subdividen de la misma forma, pero se tiene un paso extra de asignar un procesador para cada una de las partes resultantes. Este método es comúnmente llamado modelo de programa simple de múltiples datos, porque cada proceso ejecuta el mismo programa en subconjuntos diferentes. La principal ventaja de este método es que se puede lograr un alto grado de paralelismo. Las soluciones tienden a escalarse adecuadamente aumentando el número de procesadores. Cuando hay un gran número de procesadores

disponibles, este método es a menudo uno de los mejores para lograr un alto rendimiento (Chakrabarit and Mutucrhismas, 1996). Un inconveniente posible es que la dimensionalidad puede estar limitada por el costo de la comunicación entre los procesos. Para lograr el mejor funcionamiento, a menudo es importante considerar los costos de comunicación y localización de los datos entre los procesadores. En las mejores situaciones posibles, no hay dependencia entre los procesadores, y en caso de algún fallo, cada procesador está obligado a compartir información con los otros. Afortunadamente, muchos algoritmos de visualización tienen pocas dependencias de comunicación entre procesadores, por lo que el paralelismo de datos es a menudo una de las técnicas más efectivas para lograr un buen funcionamiento.

1.4 Ventajas y Aplicaciones de la Visualización Científica

La visualización científica representa una poderosa herramienta, relativamente moderna en la computación, que posibilita una representación de los datos de varias dimensiones o variables, con un nuevo enfoque, garantizando que la representación gráfica de un problema pueda efectuarse con total independencia. A través de la visualización científica las personas logran una interacción directa con los datos. La visualización puede ser hecha sin mayor dificultad en datos no homogéneos o que no se conozca detalladamente su estructura. La exploración visual es intuitiva, no requiere de complicados conocimientos matemáticos, estadísticos o de otra índole. Otra gran ventaja de la visualización es la gran cantidad de conocimiento que puede ser rápidamente interpretado, aunque consecuentemente con esto se presenta la necesidad de una gran capacidad de cómputo para obtener los resultados en un tiempo asequible.

A pesar de que en muchas de las áreas de investigación en la computación, los análisis son algunas veces más teóricos que prácticos, y que muchas veces esta teoría no llega a materializarse, la visualización científica ha surgido como una forma clara para los científicos, de asimilar, comprender y tomar decisiones de forma práctica en casi todas las ramas de la sociedad, marcando pautas en otras ciencias como la biología y la física. Procesos geográficos

y geofísicos pueden ser fácilmente comprendidos, incluso por personas que no son especialistas en el tema, gracias a las novedosas técnicas de visualización científica que permiten modelar fenómenos y objetos de los que se conoce su existencia pero que no han sido observados todavía por el hombre, como los llamados "Agujeros Negros". En el área de la medicina su repercusión ha sido también determinante, se han creado nuevos instrumentos que permiten a través de la visualización de fluidos y de volúmenes determinar enfermedades, e incluso el nivel de desarrollo de estas, por ejemplo, el crecimiento de algún tipo de células malignas que serían imposibles de diagnosticar solamente con los datos. También en el área de la comunicación, la visualización científica ha encontrado su lugar, un ejemplo común está en los programas del estado del tiempo los cuales combinan datos atmosféricos con colores y otra serie de parámetros específicos que permiten transmitir de forma clara y precisa a las personas la información meteorológica. En la química se aprecian varios usos como la representación de moléculas y estructuras atómicas. Otras aplicaciones importantes se encuentran en la rama de la vulcanología, como la visualización de la actividad sísmica en un área determinada. Debido a que la visualización científica ha representado un hito en la computación, muchas instituciones de investigación científica, centros de ciencias y organizaciones han girado sus aplicaciones para prestar servicios de este tipo. En el epígrafe posterior se realiza un estudio de cómo se implementan y se gestionan los datos científicos con este enfoque.

En el resto de este capítulo se tratan algunas implementaciones sobre las cuáles es soportado un sistema de gestión de la información en las bases de datos científicas.

1.5 Tecnologías para el almacenamiento de datos científicos

Los archivos secuenciales, sin lugar a dudas, resultan eficientes para el almacenamiento de datos científicos y permiten procesarlos y almacenarlos de forma relativamente sencilla.

Sin embargo, los científicos han demandado para su actividad un método eficiente e independiente para la lectura y escritura de dichos datos, que permita desarrollar bibliotecas

específicas para el almacenamiento. Una de las implementaciones más usadas han sido los archivos binarios con formatos estándares. Estos formatos permiten representar arreglos y datos tabulares, así como las interrelaciones y el almacenamiento de información sobre ellos, llamados metadatos, mediante la inclusión de textos descriptivos, definiciones y comentarios. Como ejemplos de estos formatos estándares se encuentran NetCDF, HDF y FITS (Li et al., 2003).

Estos formatos incluyen estructuras para soportar grandes cantidades de información, arreglos multidimensionales, una amplia variedad de tipos de datos multivariados, multievaluados y métodos para la gestión de metadatos.

Entre los estándares antes mencionados el que ha sido de más utilidad para esta investigación es el HDF. En este epígrafe se trata de forma específica este formato de datos científicos, profundizando en su estructura y las funcionalidades de su biblioteca de funciones.

1.5.1 Hierarchical Data Format (HDF)

HDF es un formato de datos desarrollado por el Centro Nacional de Aplicaciones de Súper cómputo (NCSA) de la Universidad de Illinois y se encuentra libremente disponible.

Este estándar presenta varios niveles de interacción (Ver Figura 11.7). En su nivel más bajo, HDF es un formato de archivo físico para el almacenamiento y en su nivel más alto es una colección de servicios y aplicaciones para manipular, visualizar y analizar la información en los ficheros de su tipo (Keim, 2002).

Entre estos niveles existe una biblioteca de programas que provee una interfaz de programación de aplicaciones de alto nivel y una interfaz de datos de bajo nivel.

Figura 1.7 Niveles de interacción en HDF

De forma general puede plantearse que HDF está integrado por:

1. Una biblioteca de funciones disponible en C y FORTRAN 77.
2. Herramientas de análisis y visualización científica que leen y escriben ficheros HDF.
3. Servicios de línea de comandos para operar directamente en los ficheros HDF.

Entre los servicios para el trabajo con ficheros se encuentran:

- Conversión de un formato a otro (por ejemplo, desde y hacia JPEG).
- Análisis y visualización de datos en ficheros HDF.
- Manipulación de ficheros HDF.

Puede destacarse que dentro de las posibilidades de este formato, 'hdp' es una de las herramientas más importantes pues provee información rápida sobre el contenido y los objetos de datos en un fichero HDF. Permite listar el contenido de un archivo en varios niveles para realizar un análisis más profundo, así como vaciar los datos de uno o más ficheros, en formato binario o ASCII.

Las API's incluyen conjuntos de rutinas para el almacenamiento y acceso a tipos de datos específicos donde todos los detalles de bajo nivel pueden ser excluidos. Están divididas en dos categorías: las interfaces multiarchivo y las interfaces de un sólo archivo (y).

Tabla 1 Interfaces multiarchivo de programación de HDF

API's multiarchivo	Tipo de datos que soporta
SD API	Almacena, administra y recupera arreglos multidimensionales de caracteres o datos numéricos junto con sus dimensiones y atributos, en más de un archivo.
VS API	Almacena, administra y recupera datos multivariados almacenados como un registro en una tabla.
V API	Crea grupos de cualquier estructura de datos primaria HDF.
GR API	Almacena, administra y recupera imágenes tipo raster con sus dimensiones y colores en más de un archivo.
AN API	Almacena, administra y recupera el texto usado para describir un archivo o cualquiera de las estructuras de datos contenidas en un archivo. Puede operar con varios archivos a la vez.

Tabla 2 Interfaces de un solo archivo de programación de HDF

API's de un sólo archivo	Tipo de datos que soporta
DFR8 API	Almacena, administra y recupera imágenes raster de 8 bits con sus dimensiones y paletas en un archivo.
DF24 API	Almacena, administra y recupera imágenes de 24 bits y sus dimensiones y colores en un archivo.
DFP API	Almacena y recupera colores de 8 bits en un archivo.
DFAN API	Almacena, administra y recupera cadenas de texto usadas para describir un archivo o cualquiera de las estructuras de datos

	contenidas en el archivo.
DFSD API	Almacena, administra y recupera en un archivo, arreglos multidimensionales de enteros o datos en punto flotantes, junto con sus dimensiones y atributos.

La estructura básica para el encapsulamiento de datos contiene un encabezamiento que identifica un archivo HDF y al menos un bloque descriptor de datos formado por un número de descriptores como se muestra en la siguiente figura.

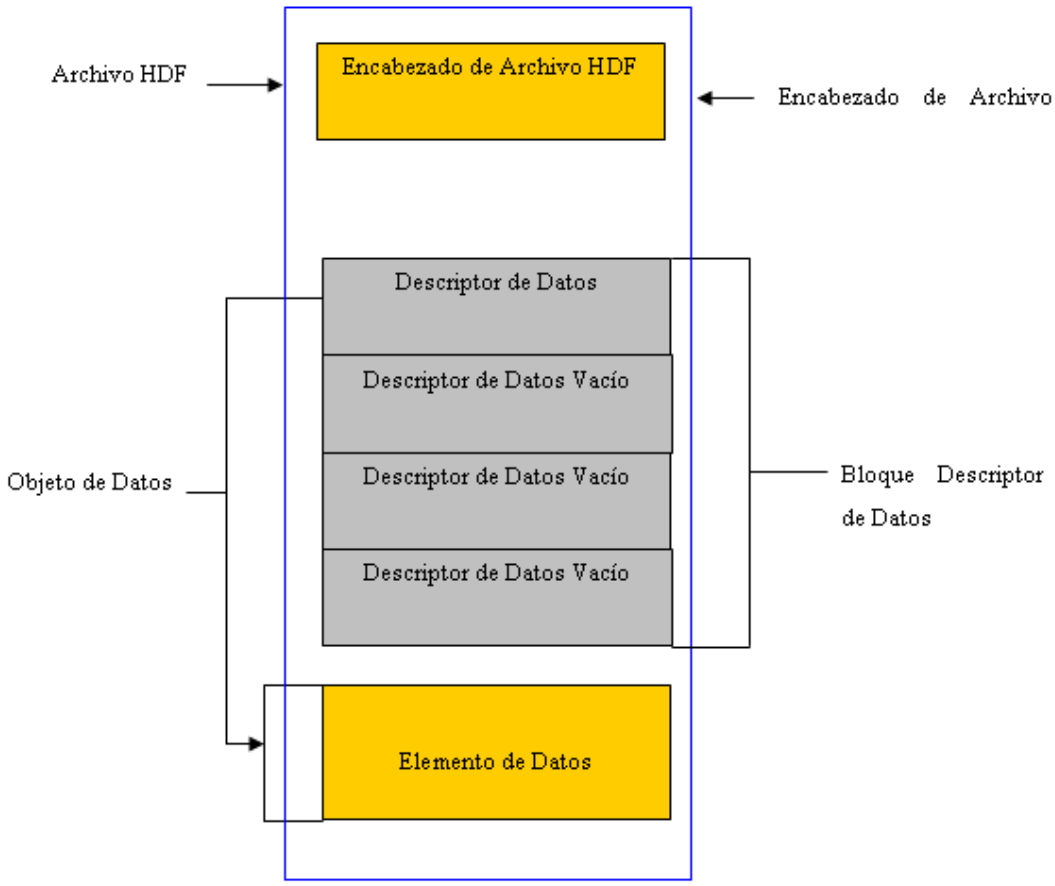

Figura 1.8 Esquema físico de un archivo HDF que contiene un descriptor de datos

Los archivos HDF contienen una serie de rasgos de gran importancia:

- La versatilidad, posibilitando el soporte de HDF a varios de modelos de datos. Cada modelo define un conjunto específico de tipos y provee una API para lectura, escritura y organización de los datos y metadatos del tipo correspondiente. Estos modelos incluyen arreglos multidimensionales, imágenes de puntos y tablas.
- La auto-descriptividad, permitiendo que una aplicación pueda interpretar la estructura y contenidos de un fichero sin ninguna información proveniente del exterior.
- La flexibilidad, permitiendo mezclar y asociar objetos relacionados agrupados en un fichero y acceder a ellos como un grupo o como objetos individuales. También permite a los usuarios crear sus propias estructuras de agrupamiento utilizando un atributo de HDF llamado *vgroup*.
- La extensibilidad, que permite acomodar fácilmente nuevos modelos de datos, sin verificar quién los adicionó, el equipo de desarrolladores de HDF o sus usuarios.
- La portabilidad, que facilita el compartimiento de los archivos HDF a través de la mayoría de las plataformas comunes, incluyendo varias estaciones de trabajo y computadoras de alto desempeño. Un archivo HDF creado en una computadora puede ser leído en un sistema diferente sin modificación alguna.

Estos rasgos hacen posible que los servicios de HDF sean factibles cuando se trabaja con suficiente información sobre un conjunto de datos, generando, por ejemplo, imágenes que representen la elevación en cada punto, como se muestra en la Figura 1.9.

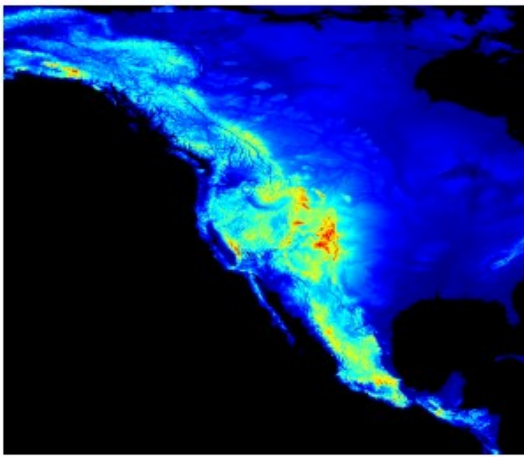

Figura 1.9 Imagen generada con los servicios de HDF

Con el desarrollo de la visualización científica, las implementaciones de HDF han evolucionado, la distribución actual es llamada HDF5. Desde versiones anteriores se ha podido estudiar que sus diferencias están dadas fundamentalmente por el formato de archivo, sus nombres, así como el modelo de los datos (Alsberg et al., 2003). Es importante señalar también que en versiones anteriores no se había logrado el soporte de la programación MPI (*Message Passing Interface*) que representa la principal atracción de HDF5.

1.5.2 HDF5 Paralelo

La característica fundamental de esta tecnología es el trabajo con plataformas paralelas y una interfaz de Entrada-Salida paralela portátil a diferentes plataformas. Este estándar tiene como meta inicial el soporte de la programación MPI pero no la programación de memoria compartida pues tuvo que ser diseñado sobre la idea del trabajo con un archivo único accedido por todos los procesos causando un procesamiento costoso (Haining, 2000).

Este formato permite que una aplicación lea de un directorio lleno de archivos en lugar de sólo un archivo. Los archivos de HDF5 Paralelo son tratados de manera muy similar a los directorios. Es necesario antes de crear un archivo, asegurarse de que el directorio sobre el cual se va a trabajar esté creado, de no ser así, es necesario hacerlo facilitando el trabajo con dicho archivo desde cualquier lugar del sistema sin pérdida de información. Los lenguajes de programación que comúnmente son utilizados para implementar las aplicaciones de HDF5 Paralelo son Fortran, C y C++.

HDF5 Paralelo está compuesto por diferentes capas, como muestra la Figura 1.10.

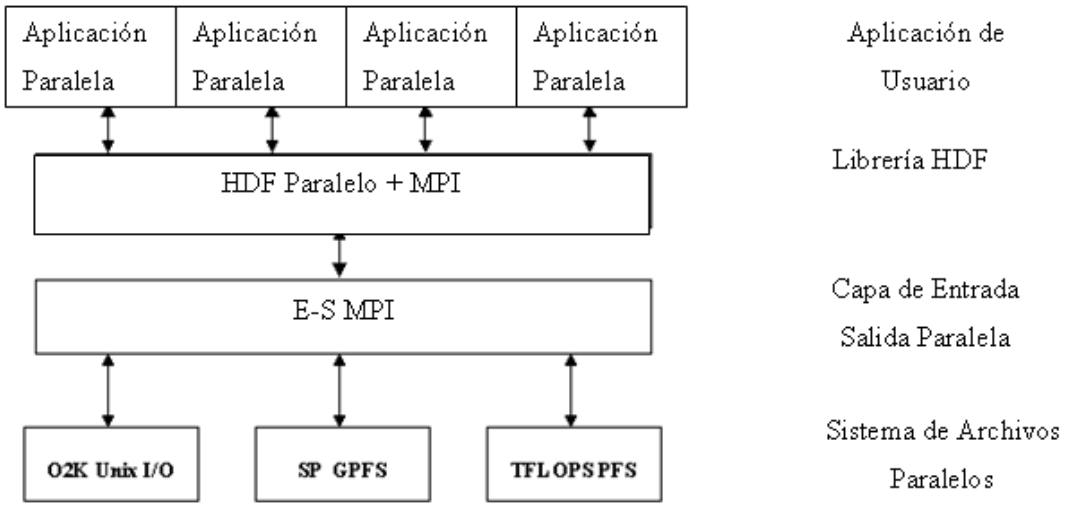

Figura 1.10 Capas de HDF5 Paralelo.

Como se mencionó anteriormente, HDF5 Paralelo está basado en el trabajo con archivos, lo que es posible mediante API's de colectividad:

- Operación con archivo: Permite trabajar con archivos en la creación, apertura y cierre del mismo.
- Creación de objeto: Permite la creación, apertura, y cierre de un *dataset*.
- Estructura de objeto: Permite obtener la extensión de un *dataset*.
- Operaciones de *dataset*: Permite la escritura o lectura de un *dataset*.

HDF5 Paralelo abre un archivo a través de un comunicador MPI permitiendo la comunicación entre el grupo de procesos que intervienen. En cuanto un archivo es abierto por los procesos de un comunicador, se facilita el acceso a todas las partes del archivo, así como a sus objetos, por todos los procesos y la escritura de los procesos múltiples al mismo *dataset* donde cada proceso escribe a un *dataset* individual.

El trabajo con datos científicos mediante el uso de este formato es de gran utilidad por lo que se hace necesario que se sigan de forma rigurosa un grupo de pasos sencillos que permiten que esta tecnología funcione correctamente:

1. Poner un objeto de plantilla de acceso para controlar el mecanismo de acceso al archivo.
2. Abrir el archivo.
3. Cerrar el archivo.

Como parte de esta tecnología se ha pretendido incorporar el trabajo con partículas mediante la API H5Part, implementando una interfaz de datos paralela con gran rendimiento para la simulación de partículas. Su esquema de archivo es desarrollado a partir de HDF5 Paralelo logrando la simplicidad de escritura y lectura de datos y las API's están orientadas a la necesidad física de partículas. Por lo tanto H5Part hereda muchas de las características deseadas de HDF5 como plataforma independiente, el paralelismo, así como los lenguajes de programación utilizados. A través del lector de H5Part se brindan una amplia variedad de formas de visualización (Adelmann, 2006).

H5Part está compuesto por varios componentes:

- Partículas
- Bloques (Permite construir los datos estructurados)
- Topología

Los archivos de H5Part pueden ser diseñados incluyendo:

- Atributos de archivo: unidades, título, archivo de mallas.
- Atributos de paso: campos, partículas, cantidades estadísticas.

A pesar de las facilidades brindadas por estos formatos de datos científicos, no son eficientes en numerosas aplicaciones donde es necesario procesar grandes volúmenes de datos que poseen una estructura bien definida. Para este caso, las bases de datos científicas han demostrado ser de gran utilidad convirtiéndose en una tecnología consolidada de amplia aceptación. Basado en lo anterior, en el próximo epígrafe se profundizará en este tema.

1.6 Bases de Datos Científicas

Las bases de datos científicas representan herramientas fáciles y flexibles que permiten procesar datos de forma eficiente para el trabajo investigativo. Mediante el uso de esta tecnología es posible verificar el estado previo de los datos, así como la modificación de los mismos, debido a que generalmente almacenan datos históricos. Las bases de datos científicas son muy extensas, pero el tamaño por sí solo no hace distinción entre una base de datos científica y una base de datos comercial. En cambio, las bases de datos científicas parecen estar separadas de sus parientes comerciales por aspectos fundamentales (Buneman et al., 2004).

Uno de los principales aspectos que marcan esta diferencia son las entidades observadas, siendo frecuentemente más complejas con interrelaciones significativas. Consecuentemente, la tecnología orientada a objetos a menudo parece más apropiada permitiendo un mejor diseño de los datos (Arkor, 2001). También las bases de datos científicas son pocas veces orientadas a transacciones. Los datos registrados son para ser preservados con posterioridad y no están constantemente cambiando para reflejar el estado actual de un sistema. La recuperación de datos es frecuentemente volumétrica, basada en dos o más rangos conjuntivos (Stolte and Alonso, 2002).

En el Observatorio Astronómico Nacional de Llano del Hato (OAN), en los Andes Merideños, se implementó una base de datos capaz de contener información sobre el sondeo del firmamento en la franja ecuatorial. La base de datos científica, posteriormente nombrada Base de Datos de Variabilidad en Objetos Celestes (BDVar), constituye un sistema concebido e implementado sobre una base de datos relacional y representa un ejemplo clásico de cómo implementar un sistema de bases de datos que brinde información y servicios a científicos de todo el mundo fomentando el trabajo colaborativo. Aunque muchas veces este tipo de instituciones no comparten los datos sobre los cuales basan sus investigaciones, sí es posible tener acceso a los resultados e incluso realizar simulaciones con datos propios basados en sus modelos de análisis.

1.7 Bases de Datos Paralelas

Una de las implementaciones más eficientes para las bases de datos científicas, debido a su exitoso desempeño basado en su gran capacidad de cómputo, son las bases de datos paralelas. Para comprender mejor este concepto es necesario en primer lugar conocer las bases de datos distribuidas.

Una base de datos distribuida es una colección múltiple de datos lógicamente relacionada a través de una red de computadoras. Un sistema de administración de bases de datos distribuido se define como un sistema de software que permite la distribución de los datos de forma transparente para los usuarios. La forma en que se implementa la distribución es muy importante para esta investigación, pues resulta un factor determinante a la hora de gestionar los datos en esta alternativa de implementación. Una de las características principales de las bases de datos paralelas está determinada por la arquitectura sobre la cual basan su funcionamiento, que puede variar de acuerdo a la capacidad económica de la institución que la implementa.

El avanzado desarrollo de la arquitectura de computadoras ha permitido establecer una nueva generación de computadoras personales que utilicen más (y mejores) componentes integrados en un sólo sistema (Foreman and Zang, 2000). Este es el caso de aquellas computadoras que soportan el trabajo de más de un procesador simultáneamente; que se denomina actualmente procesamiento paralelo.

El soporte de varios procesadores en una misma *mainboard* ha permitido agilizar el procesamiento de operaciones por parte de las computadoras, produciendo un aumento de la eficiencia en los sistemas informáticos. La paralelización de operaciones consiste en separar una operación a procesar, en suboperaciones; cada una de las cuales es ejecutada de forma simultánea en cada procesador para luego reunir los resultados en una sola respuesta.

Esta característica de las nuevas PC´s ha permitido que las bases de datos puedan ser implantadas en sistemas de esta naturaleza, tomando el nombre de bases de datos paralelas. Como se ha mencionado, el principal interés de esta investigación es aprovechar las potencialidades de cómputo de las bases de datos paralelas sobre un diseño de datos distribuido que permita implementar una base de datos científicas donde el proceso de gestión de la información se realice de forma eficiente.

Estas bases de datos no sólo son implantadas en una máquina centralizada, pueden utilizar una red de alta velocidad que enlaza varias computadoras; todas ellas trabajando conjunta y simultáneamente para construir una base de datos paralela.

Los sistemas de bases de datos paralelas comenzaron a desarrollarse en los años 80 sobre un sistema de procesadores de propósito general funcionando en paralelo. Sobre este modelo desarrollado se basan los sistemas actuales de IMB, Oracle, Infomix entre otros. Estos sistemas están constituidos por un conjunto de procesadores y discos de almacenamiento, comunicados por una red de alta velocidad, que trabajan de forma conjunta ejecutando consultas y transacciones de los clientes. Esta forma de ejecutar las transacciones mejora considerablemente la productividad, ejecutando sub-transacciones de forma simultánea.

La arquitectura es un factor determinante para la implementación de una base de datos pues existen varios modelos para el procesamiento paralelo:

- **Memoria compartida:** Todos los procesadores comparten una memoria común. Es muy eficiente en cuanto a comunicación entre procesadores.
- **Disco Compartido:** Los procesadores comparten un conjunto de discos de almacenamiento pero no es posible trabajar con un conjunto grande de nodos. Esta arquitectura permite una mayor escalabilidad entre los nodos.
- **Sin compartimiento:** Cuando los procesadores no comparten ni memoria, ni discos. Cada nodo contiene un procesador, memoria y sus propios discos. Debido a que los

datos se transportan desde los discos hasta la memoria, dentro de un mismo nodo se minimizan las colisiones en la red. Este tipo de arquitectura es escalable a miles de nodos.

- **Jerárquico:** Cuando los procesadores comparten la memoria principal y los discos de almacenamiento.

Es importante señalar los tipos de paralelismo que se pueden encontrar en estas arquitecturas:

1. **Paralelismo de Entrada salida:** Permite la reducción del tiempo necesario para la recuperación de datos, dividiendo la base de datos entre varios discos a través de una división horizontal.
2. **Paralelismo entre consultas:** Permite la ejecución de consultas o transacciones en paralelo.
3. **Paralelismo en consultas:** Permite que una única consulta se ejecute de forma paralela en diferentes procesadores y discos.

4. **Paralelismo entre operaciones.**
    a) **Paralelismo de Entrecruzamiento:** Permite ejecutar varias operaciones sin necesidad de acceder al disco para almacenar datos intermedios
    b) **Paralelismo Independiente:** Cuando se ejecutan varias operaciones independientes que pertenecen a la misma consulta.

La arquitectura de sistema de base de datos paralela es basada fundamentalmente en un *hardware* sin compartimiento donde los procesadores se comunican entre sí enviando mensajes mediante una red de interconexión (Ver Figura 1.11). En dichos sistemas, las tuplas de cada relación son particionadas (*declustered*) al otro lado del disco por unidades de almacenamiento fijando cada procesador (DeWitt and Gray, 1992). Con el particionamiento se permite que múltiples procesadores escaneen grandes relaciones paralelas sin necesitar cualquier

dispositivos de Entrada-Salida exóticos. Dicho diseño es utilizado actualmente por Teradata, Tandem, NCR, Oracle-nCUBE, y por algunos otros productos de bajo desarrollo. La comunidad de investigación también ha abarcado esta arquitectura de *dataflow* sin compartimiento en sistemas como Arbre, Bubba y Gamma.

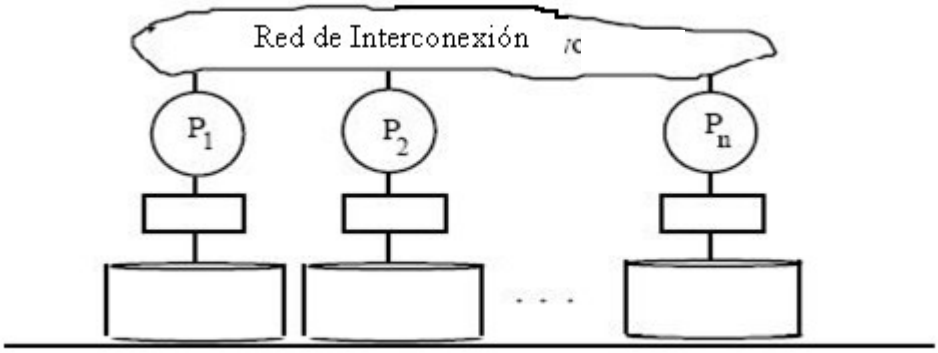

Figura 1.11 Arquitectura sin compartimiento de bases de datos paralelas

Los sistemas de bases de datos paralelas presentan importantes ventajas con respecto a otros sistemas pues permiten un aumento en la velocidad, necesitando menor tiempo de ejecución para cada transacción. También se logra la ampliación, al poder procesar tareas más largas en el mismo tiempo. A pesar de estas posibilidades que brindan las bases de datos paralelas, el costo de inicio es alto en el momento de la instalación y puesta a punto del sistema (Haddelton, 2006).

A pesar de parecer similares, los sistemas de bases de datos paralelas se diferencian de las distribuidas ya que estas se encuentran generalmente en diferentes puntos geográficos, se administran de forma separada y poseen una interconexión más lenta. También, en los sistemas distribuidos se dan dos tipos de transacciones: locales y globales. Las transacciones locales son aquellas que se ejecutan solamente en el nodo donde se inicia la transacción, mientras que las globales involucran intercambio de datos entre los distintos nodos que conforman el sistema.

En un sistema paralelo todas las transacciones tienen el mismo nivel, ya que todas son atendidas por el sistema en conjunto (independientemente de ser paralelizadas o no). La arquitectura de bases de datos sin compartimiento es similar a un sistema distribuido interconectado por holgura, a diferencia de las bases de datos distribuidas que asumen la holgura en la interconexión con su sistema operativo. Las bases de datos paralelas trabajan con arquitectura de computadoras multiproceso posibilitando un alto rendimiento y alta disponibilidad para servidores de bases de datos con menor costo.

De forma general, puede afirmarse que las bases de datos paralelas agilizan el proceso de consultas a un SGBD a través de la ejecución de forma simultánea de las transacciones, sin embargo, el costo de la implementación es mucho mayor que otros tipos de bases de datos. Los sistemas de bases de datos paralelas están teniendo una gran aceptación comercial, pero sólo en organizaciones que tienen complejos sistemas.

1.6.1 Metodología de diseño de Bases de Datos Paralelas

La metodología de diseño de una base de datos paralela es similar a la de una distribuida, diferenciándose solamente en la forma de ubicar los fragmentos. Por lo tanto, el diseño puede resumirse en los siguientes pasos:

- Diseño lógico de los datos.
- Diseño de la distribución de los datos.

Existen diversas formas de afrontar el problema del diseño de la distribución:

- Los procesos fundamentales: la fragmentación y la asignación, se abordan de forma simultánea. Esta metodología se encuentra en desuso.
- Los procesos fundamentales asumen un enfoque en dos fases: primeramente se realiza la partición para luego asignar los fragmentos generados. Esto se debe a dos razones: primero, para manejar mejor la complejidad de diseño y segundo, porque es relevante desde el punto de vista conceptual ya que la fragmentación trata los criterios lógicos que

motivan la división de una relación global mientras que la distribución tiene que ver con la ubicación física de los datos en los diferentes nodos de la red. A continuación se comentan los aspectos fundamentales de los pasos del diseño de distribución.

1.7.1.1 Fragmentación de los datos

Existen varias alternativas de fragmentación determinadas por la forma de particionar los datos (horizontal o vertical). La fragmentación horizontal está dada por una operación de selección y la vertical por una operación de proyección.

La fragmentación vertical consiste en la división de una relación en fragmentos, cada uno de los cuales contiene un subconjunto de los atributos de la relación original. Su principal objetivo es mejorar el rendimiento de las transacciones; para esto los fragmentos deben ser diseñados de tal manera que se adapten a las necesidades de las aplicaciones, es decir, que estas accedan al menor número de atributos irrelevantes posibles. Se entiende por atributos irrelevantes aquellos que no son necesarios para resolver la consulta. Además, la fragmentación vertical permite aumentar el paralelismo durante la ejecución de las consultas, la concurrencia y el rendimiento. La división o fragmentación horizontal trabaja sobre las tuplas, dividiendo la relación en subrelaciones que contienen un subconjunto de las tuplas que alberga la primera.

Esta fragmentación puede ser implementado de distintas formas como son *Round-robin y Hash function,* diferenciándose en cuanto al tamaño del índice.

- *Round-robin*: Es utilizada cuando se quiere acceder a la relación escaneando todo secuencialmente sobre cada pregunta, pero es deficiente cuando se desea acceder a tuplas, queriendo decir que la aplicación quiere encontrar todas las tuplas conjuntamente y obtener un valor de atributo especial.
- *Hash function*: Es utilizada cuando se desea el acceso a los datos de manera secuencial y asociativa. Es la más recomendada en el trabajo con cluster de

computadoras.

La asignación de los datos de forma paralela permite su colocación en el lugar exacto donde van a ser usados. También favorece el acceso en paralelo permitiendo la participación concurrente de varios procesadores en la ejecución de una consulta.

1.7.1.2 Asignación de los datos

Asumiendo que la base de datos está adecuadamente fragmentada, se procede a ubicar los fragmentos en los diferentes sitios de la red. Cuando los datos son ubicados pueden estar replicados o mantenidos como una sola copia. En el caso de este trabajo los datos serán replicados posibilitando la confiabilidad y la eficiencia de las solicitudes de sólo lectura. Si existen múltiples copias de un elemento de datos habrá una mayor probabilidad de que alguna copia esté accesible en algún lugar, incluso si ocurriera una falla en el sistema. Mejor aún, las solicitudes de sólo lectura que acceden a los mismos artículos de datos pueden ejecutarse de forma paralela ya que las copias existen en múltiples sitios.

1.6.2 Funcionalidades de Bases de Datos Paralelas

Dentro de las principales características de las bases de datos paralelas se encuentran: el control de concurrencia, los protocolos de recuperación, los protocolos de replicación, entre otras (Silberschatz and Korth, 1998).

El control de concurrencia se lleva a cabo ante la necesidad del sistema de controlar la interacción entre las transacciones concurrentes. Esto sucede cuando se ejecutan al mismo tiempo varias transacciones en la base de datos, donde puede perderse la consistencia de la misma. La sincronización de las transacciones concurrentes puede extenderse a un argumento de seriabilidad. En sistemas donde las operaciones de una transacción en particular pueden ejecutarse en sitios múltiples donde se acceden a los datos se requiere un seriabilidad global.

Con esto se pretende que las transacciones en cada sitio sean serializables y además, que el orden de serialización en todos los sitios sea idéntico.

El esquema de recuperación es una parte integrante del sistema de bases de datos, el cual es responsable de la detección de fallos y del restablecimiento de un estado de la base datos anterior al momento de producirse, lo cual se denomina protocolo de recuperación. Un fallo puede estar dado por varios motivos, incluyendo fallos de discos, cortes de corriente o fallos en el software. En cada uno de estos casos puede perderse información concerniente a la base de datos (Shankar, 2007). Las transacciones pueden fallar, además de por un fallo del sistema, por otras razones, como una violación de las restricciones de integridad o interbloqueos. En los sistemas distribuidos, la recuperación de datos hace cumplir la atomicidad en las transacciones, mediante la implementación de protocolos de reclusión atómicos.

En la replicación de bases de datos distribuidas cada ítem de dato lógico tiene varios ejemplos físicos. El asunto, en este tipo de sistema de base de datos, es mantener la noción de consistencia entre las copias de instancias físicas cuando los usuarios pongan al día sus artículos. Un protocolo típico de control de réplica hace cumplir una copia serializable sobre la lectura de uno y la escritura de todos, conocido como ROWA. Este protocolo es simple y sencillo, pero requiere de todas las copias de todos los ítems de los datos, lo cual es actualizado por una transacción, permitiendo el acceso a la misma al terminarse. El fracaso de un ítem puede bloquear una transacción reduciendo así la disponibilidad de la base de datos.

1.8 Conclusiones Parciales

Según se ha podido analizar en el transcurso de este capítulo, el almacenamiento y gestión de los datos científicos es un problema complejo, dados los grandes volúmenes de datos que se manejan y la heterogeneidad de los mismos. En la actualidad existen nuevas y avanzadas técnicas y herramientas para este fin, que pueden ser aplicadas en el área de la visualización científica. El enfoque tradicional ha sido el uso de archivos binarios secuenciales planos, que

brindan una solución eficiente desde el punto de vista de acceso a los datos. Sin embargo, otras tecnologías de almacenamiento han evolucionado a la par, aprovechando las ventajas que brindan los SGBD, en especial sobre arquitecturas paralelas.

La gestión de datos científicos se distingue por ciertas características, entre ellas pueden relacionarse: los diferentes y complejos tipos de datos que se manejan, la complejidad de las entidades observadas, el bajo nivel de transaccionalidad y la importancia de la recuperación de datos. Es posible utilizar bases de datos paralelas para modelar este tipo de datos, teniendo en cuenta estas características propias, siempre que escoja la arquitectura adecuada y se apliquen correctas metodologías de diseño e implementación.
Estas conclusiones permiten plantear las siguientes hipótesis de investigación para este trabajo:

1. Es posible incorporar datos científicos en un SGBD sin perder la eficiencia que ofrecen los formatos nativos de almacenamiento de datos empleados en la visualización científica.

2. Existen SGBD basados en software libre que son apropiados para incorporarles datos científicos como uno de sus tipos primitivos de datos.

3. Es posible implementar la estructura de los datos científicos en una base de datos paralela.

4. Resulta eficiente incorporar datos científicos en una base de datos paralela con respecto a las soluciones existentes en la actualidad, que emplean formatos nativos para el almacenamiento de la información.

Capítulo 2Tecnologías para la implementación de Bases de Datos Paralelas sobre un cluster de computadoras.

2.1 Introducción

En numerosas aplicaciones es necesario procesar grandes volúmenes de datos que poseen una estructura bien definida. Durante mucho tiempo, las bases de datos relacionales han demostrado ser de gran utilidad convirtiéndose en una tecnología consolidada de amplia aceptación, donde los datos se representan como un conjunto de tablas que organizan la información de una forma específica, evitando problemas tales como redundancia o inconsistencia. Sin embargo, como la cantidad de información a manejar en la actualidad ha tomado dimensiones considerablemente mayores por ejemplo: gran cantidad de tablas, cada una con gran cantidad de tuplas, los tiempos de ejecución requeridos para procesar consultas de recuperación de información a la base de datos pueden fácilmente llegar a ser intolerablemente altos en computadores secuenciales (Takhur and No, 2000).

Las bases de datos paralelas ante esta situación emergen como una solución eficiente para el almacenamiento y gestión de grandes volúmenes de datos. Existen múltiples aplicaciones que permiten implementar este tipo de sistemas, pero muchos de ellos están en la categoría de software propietario, y presentan altos costos de adquisición, se puede tomar como referencia al sistema Oracle, uno de los más utilizados recientemente, pero con precios muy elevados (Karavanic and May, 2005).

Con el advenimiento de Internet, el software libre se ha convertido en una alternativa técnicamente viable y económicamente sostenible frente al comercial, contrariamente a lo que a menudo se piensa, convirtiéndose en otra nueva opción para ofrecer los mismos servicios a un costo significativamente reducido (Juanjo, 2005). El contenido fundamental de este capítulo está orientado a encontrar alternativas en el mundo del software libre para el trabajo con bases de

datos paralelas y realizar una comparación entre ellas, a partir del establecimiento de las características deseadas en un sistema de esta naturaleza. Como parte de esta investigación se ha estudiado un SGDB comercial con el objetivo de establecer pautas que sean decisivas en el momento de seleccionar la herramienta a utilizar.

2.2 Sistema Oracle RAC (Real Application Cluster)

La herramienta que brinda Oracle para el trabajo en clusters de computadoras es llamado *Oracle Real Application Clusters* (RAC), una configuración de bases de datos que presenta múltiples instancias de la aplicación con acceso compartido al mismo conjunto de archivos de datos. La configuración de RAC está dada por combinaciones de *hardware* y software específicos capaces de soportar un entorno de cluster. Oracle RAC utiliza una arquitectura de *caché* compartida, superando a las arquitecturas tradicionales ya conocidas como son: sin compartimiento y compartimiento de disco. De esta manera suministra una solución de base de datos escalable y disponible para todas las aplicaciones de empresas (Edward and Fetrow, 2007).

El funcionamiento de un cluster implementado en un ambiente de RAC permite difundir la carga en servidores múltiples que se encuentran dentro del sistema, en lugar de hacerlo en un único servidor donde dicha carga de aplicación no es soportada.

2.2.1 Arquitectura de Oracle RAC

En la arquitectura de Oracle RAC, las peticiones a la base de datos son generadas por la aplicación, por ejemplo, desde un grupo de conexiones configurado en un servidor de aplicación, y Oracle RAC en su conjunto es el encargado de direccionar las peticiones al servidor que esté en funcionamiento. Esta configuración no posee balanceo de cargas, por lo que la configuración mostrada es exclusivamente para fallos (es decir, todas las peticiones

llegarán al nodo 1, y sólo en caso de que éste deje de funcionar, las peticiones se redireccionarán al nodo 2), como se muestra en la Figura 2.1.

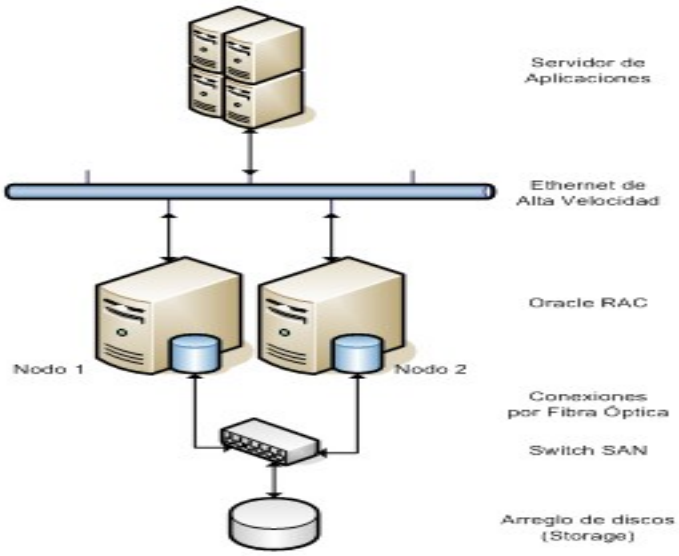

Figura 2.1 Diagrama conceptual del funcionamiento de un Oracle RAC

En la figura 2.2 se muestra de forma detallada, el funcionamiento de Oracle RAC, teniendo en cuenta la descripción de cada uno de sus componentes.

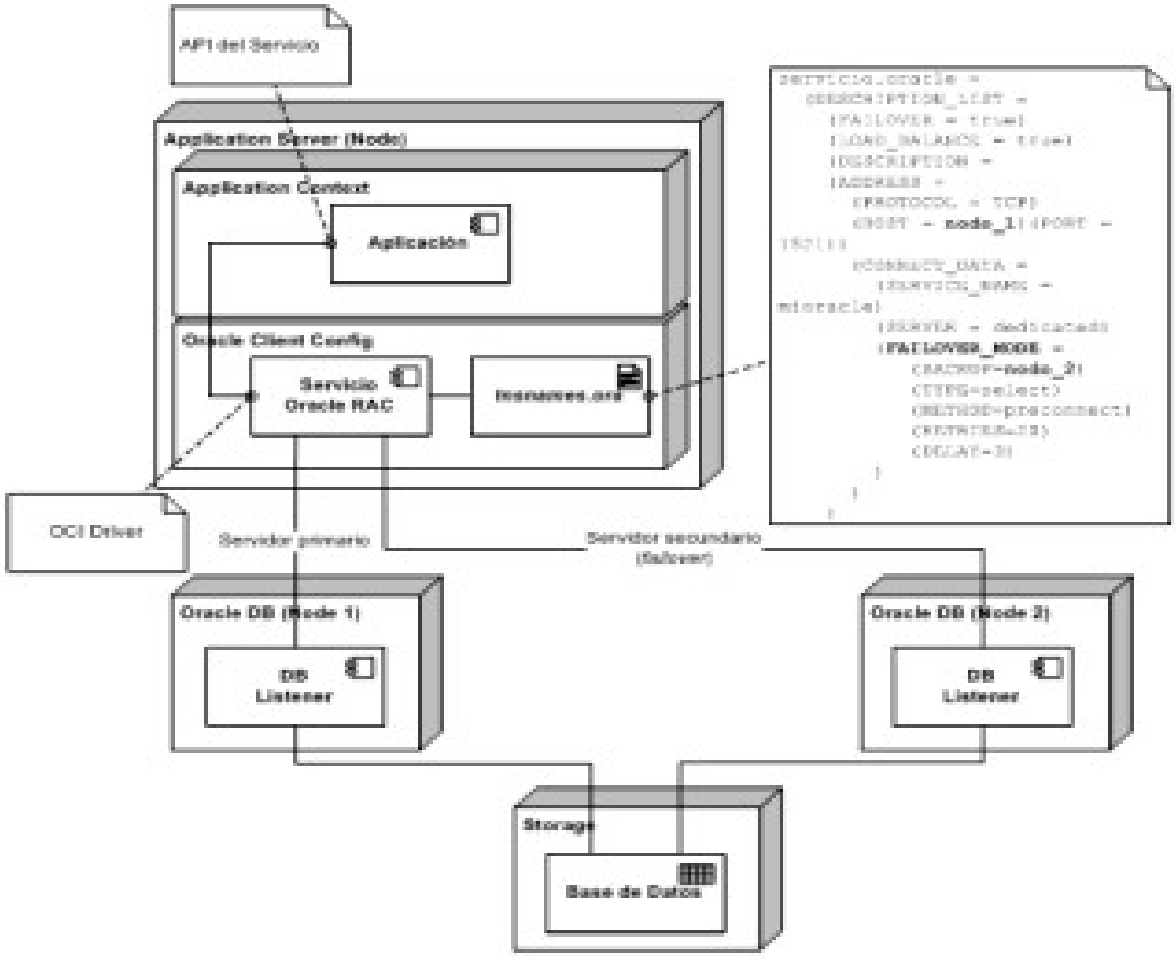

Figura 2.2 Diagrama de componentes Oracle RAC

La figura 2.2 muestra cómo el cliente o servicio de Oracle está configurado para tener como conexión primaria al nodo 1. Si no es posible ejecutar la petición enviada a dicho servidor, el servicio se encarga de realizar un redireccionamiento de la misma hacia el servidor de respaldo (en este caso el nodo 2). Adicionalmente, es necesario mencionar que lo que se está ejecutando en los nodos 1 y 2 es el agente motor de base de datos denominado *listener*, no la base de datos en sí, los archivos que componen dicha base de datos se encuentra en un arreglo de discos con configuración en espejo para proveer redundancia y por lo tanto, alta disponibilidad y fiabilidad (Wolf, 2002).

El balanceo de carga así como el proceso de fallo transparente, pueden resolverse mediante un cluster con balanceo de carga por *hardware* o software. La escalabilidad de dicha solución depende más bien del presupuesto con el que se cuenta, pero soporta la decisión de implementar Oracle RAC para alta disponibilidad de la base de datos (Pfaltz, 2000).

2.2.1.1 Oracle RAC con Balanceo de Carga

En Oracle RAC con balanceo de carga, se adiciona un nuevo componente, el facultado para realizar la distribución de carga es el balanceador, como se muestra en la figura 2.3. Éste puede ser un componente de software (por ejemplo, un *Web Server* con balanceo *Round-robin* como Apache) o uno de *hardware* (como un switch F5), lo que posibilita que no exista dependencia hacia un cliente de Oracle instalado en el servidor de aplicaciones. Esto es especialmente importante cuando no se puede instalar este componente o es necesario incrementar la portabilidad de la plataforma. A su vez se logra la optimización de los recursos al distribuir la carga entre dos o más servidores que componen la solución. El componente de balanceo es el encargado de detectar las caídas de los nodos de la capa de *back-end*, que depende del componente utilizado.

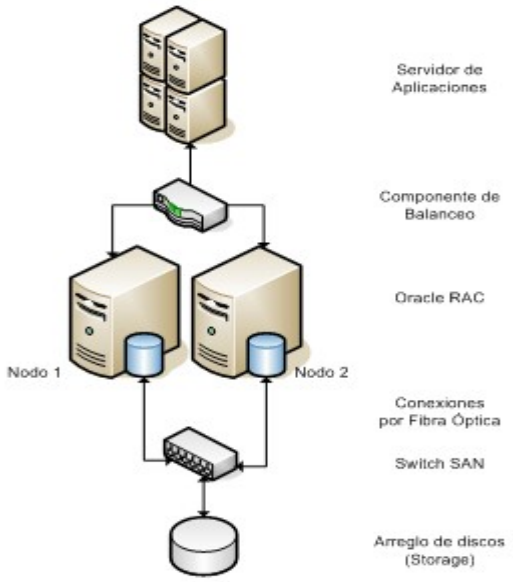

Figura 2.3 Oracle RAC con balanceo de carga

2.3 Características de las bases de datos en Oracle RAC

Esta investigación basa su atención en las alternativas de software libre para una posible implementación, sin embargo es importante tener en cuenta algunas características que hacen de los SGBD comerciales con extensiones para arquitecturas paralelas, herramientas potentes capaces de brindar un servicio profesional que no están al alcance de países o instituciones en desarrollo.

En un sistema implementado en Oracle RAC, cuando ocurre un fallo en las instancias a la base de datos, el acceso a las mismas no se pierde, solamente un subconjunto de usuarios, aquellos relacionados con el nodo que falle, es afectado. Si una instancia falla, en una base de datos RAC, es reconocida por otra instancia en el cluster y se recupera tomando automáticamente su lugar. Dentro de Oracle RAC se encuentra un subsistema llamado *Clusterware* mediante el cual se suministra una solución completa para la aplicación de base de datos, dirigiendo todos los procesos automáticamente y encargándose de monitorear el sistema de Oracle reiniciándolo automáticamente.

Este sistema provee a las aplicaciones de la habilidad de recibir inmediatamente la notificación de los componentes del cluster que fallan y de hacer transparente el fallo para el usuario, reenviando la transacción a un nodo sobreviviente en el cluster, esto se logra a través de un complejo sistema llamado *Fast Application Notification* (FAN) implementado con este fin. De forma general el FAN es capaz de manejar los escenarios de fallo de forma transparente para la mayoría de los usuarios, presentando un esquema de alta disponibilidad, que hace de Oracle RAC una alternativa a tener en cuenta, sin embargo el costo de este producto es un inconveniente determinante que hace imposible su utilización como parte de este trabajo.

Oracle implementa un control de concurrencia multiversión que difiere un tanto de los mecanismos de concurrencia utilizados por otros fabricantes de bases de datos. Soporta la consistencia de lectura en un nivel de instrucción y de transacción a la vez que determina el

número de cambio del sistema (*System Change Number*, SCN) actual. El SCN esencialmente actúa como una marca donde el tiempo se mide en términos de compromisos de la base de datos en lugar del tiempo de reloj. En el modelo de concurrencia de Oracle las operaciones de lectura no bloquean las operaciones de escritura y viceversa, esta característica permite alto grado de concurrencia.

Después de realizar un estudio sobre las principales características de Oracle RAC, y como parte de esta investigación se ha decidido tomar como puntos de referencia para la selección de una herramienta de software libre los siguientes puntos:

1. Control de Concurrencia.
2. Protocolos de Recuperación
3. Protocolos de Replicación.

Debe destacarse además que los sistemas tradicionales han tenido que incorporar herramientas y lenguajes para lograr la implementación de un funcionamiento paralelo y en específico de estas características, aspecto en el se profundizará en el resto del capítulo. Estas herramientas se dividen en dos clasificaciones, las basadas en replicación y las basadas en distribución.

2.4 Sistemas basados en replicación

Los sistemas que implementan la replicación surgen por la necesidad de tener herramientas que realicen de forma automática copias de seguridad o actualizaciones en diferentes zonas geográficas. Estos sistemas posibilitan replicar los datos en nodos diferentes, facilitando la recuperación de información en caso de pérdida de un nodo (Pacciti and Özsu, 2007). Entre los sistemas que brindan herramientas para la replicación de los datos se encuentra PostgreSQL, con extensiones para el trabajo con bases de datos paralelas tales como Slony-I, Postgres-R, CyberCluster y PGCluster.

2.4.1 Slony-I

Slony-I es un sistema de replicación en cascada que brinda la posibilidad de replicar grandes bases de datos PostgreSQL a un número razonable de sistemas esclavos. De esta forma se puede implementar un sistema que preste servicios ininterrumpidamente a pesar de las fallas que puedan ocurrir en un número determinado de nodos pertenecientes a este sistema. Slony-I está diseñado para implementar grandes centros de datos y sistemas de almacenamiento secundario donde todos los nodos están disponibles en todo momento.

- **Protocolo de Replicación**

La estructura básica de esta herramienta, como se ha mencionado, está formada por un sistema *master* y uno o varios sistemas esclavos, donde cada nodo no recibe necesariamente la información del nodo *master*, sino que todos los nodos pueden ser configurados para reenviar los datos que reciben a los restantes nodos esclavos del sistema.

Existen tres ideas básicas para implementar esta característica de Slony-I: la primera es la escalabilidad, donde el nodo *master* que recibe las transacciones de actualización desde las aplicaciones clientes, solamente tiene la capacidad, en el proceso de replicación, de actualizar los datos en una determinada cantidad de nodos esclavos; el resto de los nodos reciben la replicación de otros nodos esclavos. La segunda idea, es limitar el ancho de banda requerido para un sistema de respaldo, manteniendo la posibilidad de tener múltiples sistemas esclavos remotos.

- **Protocolos de Recuperación**

La tercera idea, en el marco de los protocolos de recuperación, es la posibilidad de configurar el sistema para posibles escenarios de fallos, ya que esta herramienta no detecta un fallo de forma automática. En este caso, especialmente donde el nodo *master* es el que falla, es poco deseable

que el resto de los nodos esclavos tengan el mismo estado de sincronización. Para asegurar que un nodo esclavo sea promovido a nodo *master* cuando este falla, es necesario que no existan inconsistencias entre el nuevo nodo *master* y el resto de los nodos esclavos. El nodo que falle pudiese ser recuperado, pero si la base de datos es muy grande podría tardar mucho tiempo en esta acción.

- **Control de concurrencia**

En esta herramienta, la regularidad de los datos es mantenida usando un control de concurrencia multiversión (*Multi Versión Control Concurrency*, MVCC). Es posible que mientras se consulte una base de datos, cada transacción sólo vea una versión de la misma, sin considerar el estado en curso de la información subyacente. Esto protege a la transacción de ver los datos inconsistentes, que podían ser causados por las otras transacciones simultáneas sobre el mismo conjunto de datos, suministrando el aislamiento de la transacción para cada sesión de datos. Mediante MVCC, se evita la metodología de cierre explícita de sistemas de bases de datos tradicionales, minimizando la discusión de bloqueo para tener en cuenta el rendimiento razonable en ambientes multiusuario.

Estas características hacen de Slony-I una alternativa a tener en cuenta cuando se necesita implementar un sistema de replicación para grandes bases de datos, en los actuales centros de ciencia y sistemas de respaldo de información en la era de las comunicaciones.

2.4.2 Postgres-R

El sistema Postgres-R es una extensión de PostgreSQL que integra el control de réplica en el núcleo de un sistema de base de datos. PostgreSQL tiene como características fundamentales el soporte de procedimientos almacenados (*triggers*), las transacciones interactivas y es un sistema multiversión, donde una transacción ve sus propias actualizaciones pero no las actualizaciones de las transacciones simultáneas que pasan a ser parte también del

funcionamiento de Postgres-R. Este último puede interactuar directamente con el control de concurrencia basado en tuplas, sin necesidad de implementar su propio mecanismo de control.

- **Protocolos de Replicación**

El sistema de Postgres-R está compuesto por procesos donde el componente principal es el director de replicación. Este proceso tiene como función principal la coordinación de los mensajes, organizar y mantener actualizada la conexión al sistema de comunicación del cluster así como el proceso de los *backends*.

Los *backends* son los que manejan las transacciones, donde cada uno puede procesar solamente una tarea a la vez. Se utilizan cuando ningún usuario quiere conectarse a ningún servidor de Postgres-R, por lo que se crea un *backend* local para cada usuario. A medida que los usuarios hacen peticiones de mensajes, estos son incorporados a los *backends* y ejecutados, lo cual ha sido un trabajo costoso. Cuando se necesita una repetición de las transacciones, el director de replicación crea los *backends* remotos, los cuales no tienen una conexión con el cliente. La estructura antes explicada se muestra en la Figura 2.4. Dentro de Postgres-R se encuentra el director de comunicación, cuya tarea es establecer una interfaz basada en *sockets* entre el director de replicación y el sistema de comunicación de grupo.

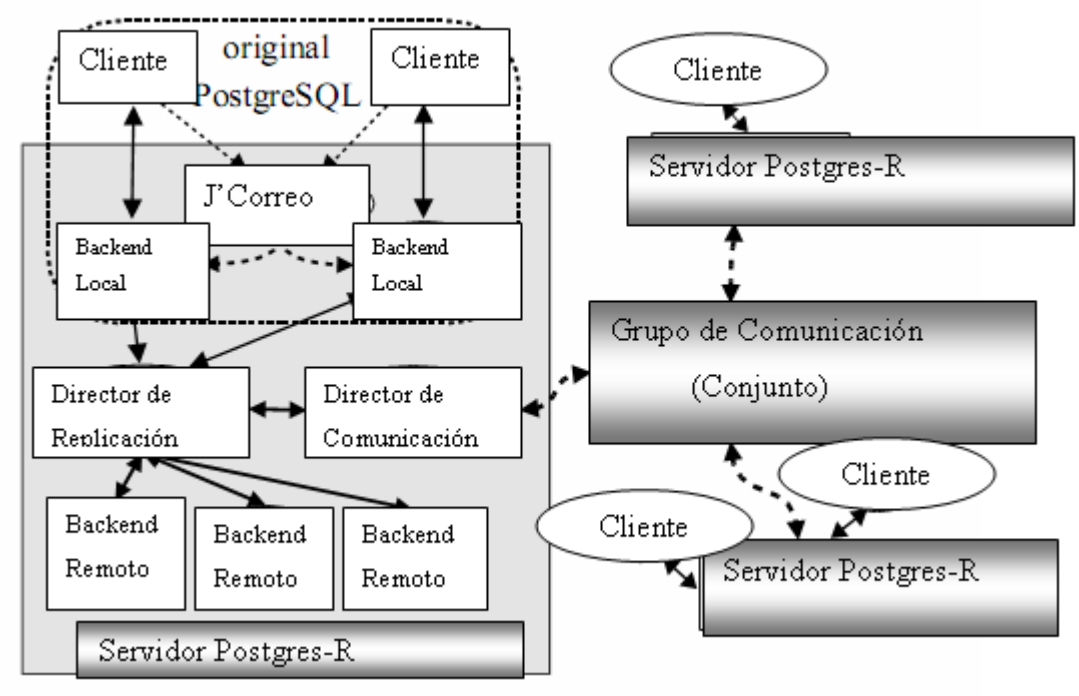

Figura 2.4 Arquitectura de Postgres-R

A continuación se muestra el funcionamiento de los componentes de Postgres-R mediante un ejemplo:

Postgres-R solamente se encarga de la operación de un nodo. La transacción escribe los datos (actualizar, insertar o eliminar comandos de SQL) y los nuevos datos son coleccionados en un conjunto de escritura (*writeset*). El *backend* local es el encargado de reunir los datos cambiados en el *writeset* hasta que recibe la solicitud del cliente. Luego, envía el *writeset* al servidor de replicación, el cual se encarga de enviarlo al resto de los nodos usando un canal totalmente arreglado por el director de comunicación, antes de devolver una respuesta. Si se desea hacer una repetición del *writeset*, es que se crea un *backend* remoto al cual se le envía dicho *writeset* y entonces es el encargado de realizar la transacción de los datos.

- **Protocolo de Recuperación**

Debido a la limitación del componente de bitácora de PostgreSQL, la actual versión de Postgres-R transfiere la base de datos completa a un nodo unión. PostgreSQL provee una función que extrae la base de datos completa en forma de un *script* y la guarda en un fichero externo, posibilitando que esta transferencia sea soportada eficientemente. Existe un nodo *master* que provee a todos los nodos unión, la versión actual de la base de datos. Sobre la unión de un nodo, el *master* ejecuta todas las transacciones desarrolladas antes de un cambio de vista, y luego ejecuta la función de extracción de la base de datos. Esta función adquiere bloqueos de lectura en todas las relaciones, y las transacciones locales en la fase de lectura pueden continuar mientras el conjunto de escritura entrante es retrasado. Un vez que el fichero es creado, los bloqueos de lectura son liberados y las operaciones de escritura pueden ser ejecutadas. El fichero es transferido a un nodo unión. Allí la base de datos es instalada y la cola de conjuntos de escritura es tratada antes que el nodo unión acepte consultas de los usuarios. Postgres-R maneja diferentes casos de fallos, reaccionado de acuerdo a si es el nodo unión o el nodo *master* el que falla durante la recuperación.

La presente distribución de Postgres-R es bastante ineficiente, ya que el nodo *master* permanece bloqueado hasta que todos los datos sean extraídos. La extracción de datos toma un par de segundos para bases de datos pequeñas (10 tablas). Esto muestra la necesidad de una gestión de bloqueos más flexible, donde los bloqueos son liberados una vez que las relaciones correspondientes son extraídas.

2.4.3 CyberCluster

CyberCluster es un sistema multi-*master* basado en PostgreSQL. La replicación se realiza de forma simultánea (sincrónica), lo cual posibilita que los datos sean guardados consecuentemente en todos los nodos de la base de datos. Este sistema se encarga de sincronizar las escrituras a la base de datos para asegurarse de que todos los nodos dentro del cluster contengan los mismos datos. A su vez posibilita que el acceso de lectura se realice de

forma balanceada, asegurándose que todos los nodos de la base de datos puedan trabajar eficientemente (Kiev, 2007).

- **Protocolo de Replicación**

El sistema está compuesto por un grupo de nodos, servidores de replicación y balance de carga. Los nodos son los encargados de procesar los pedidos. Cuando el pedido es recibido del cliente, un grupo de nodos debe determinar el tipo de pedido, si es escritura o lectura. En caso de lectura, el pedido es ejecutado por la base de datos directamente y el resultado es devuelto al cliente. Si el pedido es de escritura, el grupo de nodos envía un mensaje al director de replicación, para asegurarse de que los datos sean enviados a todos los restantes nodos de la base de datos dentro del cluster.

El servidor de replicación es el componente central del sistema, cuyo funcionamiento está dado por la toma de las solicitudes de los nodos entrantes y la reproducción de los cambios a los nodos restantes del sistema.

Mediante el balance de carga, CyberCluster distribuye la carga dentro del cluster, determinando el número de consultas activas. La máquina que contenga menos consultas activas será elegida para realizar un nuevo pedido. Si el balance de carga no está disponible, las aplicaciones no pueden conectarse a ningún nodo de la base de datos dentro del sistema. El balance de carga no es usado durante la replicación, sino solamente en la distribución de la carga.

- **Protocolo de Recuperación**

De manera automática, cuando el software del cluster detecta un error, CyberCluster separa los nodos descompuestos del sistema y continúa la operación normal sobre las bases de datos. Cuando el nodo descompuesto es reparado, puede volver a ser incorporado al sistema sin posibilidad de peligro. Esto no causa pérdida de tiempo si se logra que al menos tres nodos estén disponibles para dicha operación (un nodo activo, un nodo para sincronización y un nodo

descompuesto). En el proceso de recuperación, la base de datos del cluster se recupera de forma automática mediante un nodo *master* activo.

2.4.4 PGCluster

PGCluster es un sistema de replicación multi-*master* y simultáneo, donde las consultas son basadas en replicación. Es un programa gratuito que puede ser descargado y usado libremente bajo los términos de la licencia de BSD. Su trabajo es basado fundamentalmente sobre servidores normales, asegurando alta disponibilidad y suficiente rendimiento para la carga de lectura de los datos (Mitani, 2007).

- **Protocolos de Replicación**

PGCluster, es un sistema agrupador de datos que almacena información en un disco compartido, actuando de forma simultánea y mediante varios nodos *master*. La replicación de consultas se basa en un grupo independiente de nodos que pueden reproducirse. Esta herramienta está compuesta por un sistema de balance de carga, múltiples servidores de replicación y un grupo de bases de datos como se muestra en Figura 2.5.

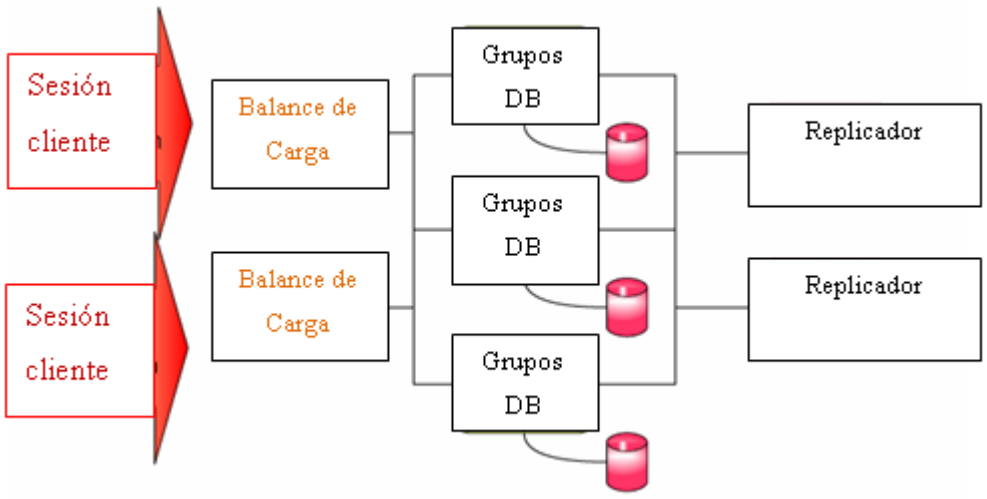

Figura 2.5 Estructura de PGCluster

- **Control de Concurrencia**

Una de las características fundamentales de PGCluster es la alta concurrencia, mediante un Control de Concurrencia Multiversión. Similar a PostgreSQL, permite que mientras un proceso escriba en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último a lo que fue comprometido. Esta estrategia es superior al uso de bloqueos por tabla o por filas, común en otros sistemas de bases de datos, eliminando la necesidad del uso de bloqueos explícitos.

Aunque PGCluster necesita un conjunto de requisitos, es necesario que el rendimiento aumente, o sea, que no sólo se permita leer, también escribir y reducir el costo de estas operaciones (Mitani, 2007). Como consecuencia de estas insuficiencias surge PGCluster-II. En esta nueva versión, como se muestra en Figura 2.6, el almacenamiento de los datos está dado fundamentalmente por disco compartido, además de que la memoria *caché* y el estado de compartimiento son tratados por *Virtual* IPC. *Virtual* IPC es el compartimiento de memoria durante el agrupamiento de los nodos, donde la escritura de nodos se realiza a través del proceso de grupo y la lectura a través del directorio del nodo local. Además, la señal y cola de mensaje están fuera de alcance.

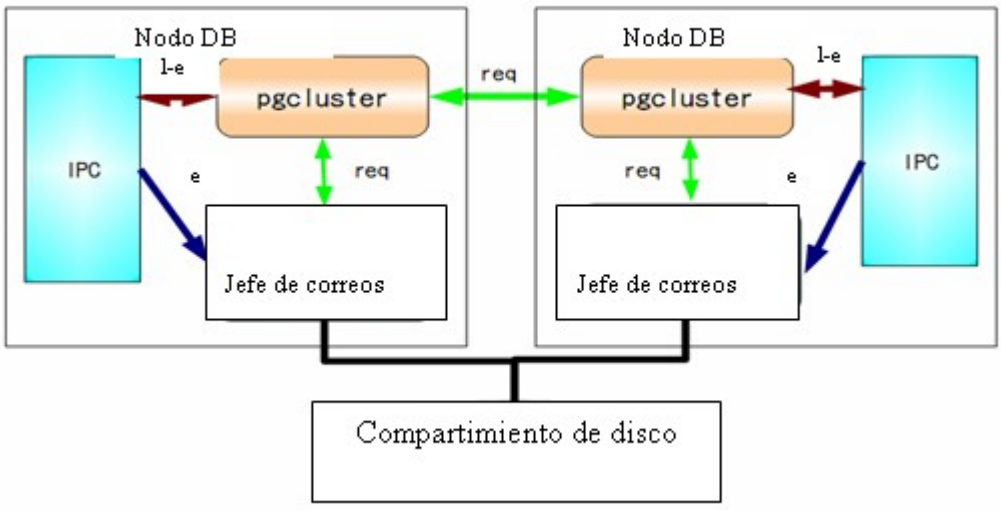

Figura 2.6 Estructura de PGCluster-II

2.5 Sistemas basados en distribución

Con el reciente desarrollo de la informática en la llamada era de la información, la necesidad de encontrar nuevas alternativas dentro de software libre para almacenar y gestionar de forma eficiente la gran cantidad de datos que es generada por los modernos instrumentos científicos y simulaciones, es un reto de las sociedades modernas (Dudoit, 2003).

Los sistemas de replicación, por la dimensión de los datos, ya no resultan una alternativa viable, abriendo paso a los sistemas basados en la distribución, que permiten de forma transparente a los usuarios dividir entre los nodos de arquitecturas paralelas la información de forma consistente y brindar un servicio más rápido a las aplicaciones. Entre los sistemas que brindan herramientas para la distribución de los datos se encuentran PostgreSQL (con Plproxy como lenguaje de manipulación de los datos) y MySQL (con MySQL-Cluster como herramienta para la distribución).

2.5.1 Plproxy

Plproxy es un nuevo lenguaje de manipulación de datos implementado para las bases de datos PostgreSQL, que permite de forma sencilla, gestionar llamadas remotas a los SGBD. Así, es posible acceder de forma transparente a los datos que se encuentran en distintos nodos del sistema. Las herramientas de distribución permiten a los usuarios dividir una base de datos entre distintos servidores. Por ejemplo, una tabla con los números de identificación de las personas de un país, que diariamente es consultada por millones de usuarios, funciona de forma más eficiente si los datos son divididos de acuerdo a algún criterio de selección como se muestra en la Figura 2.7

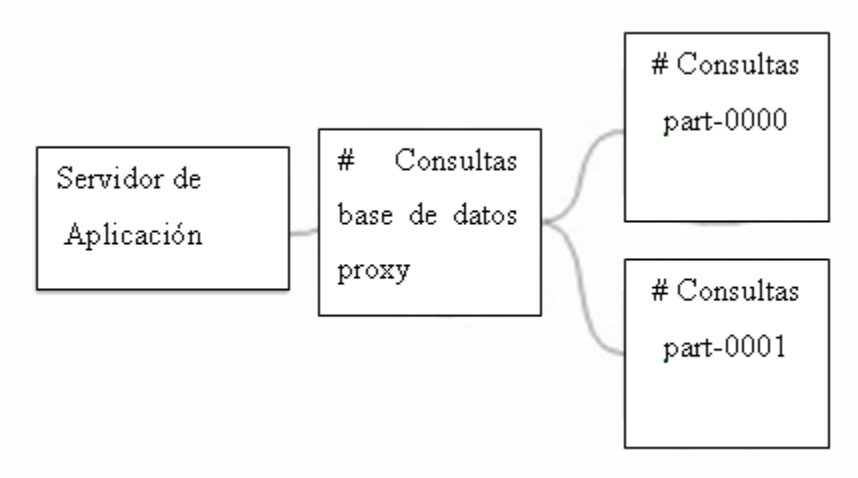

Figura 2.7 Particionamiento en Plproxy.

Luego de dividir los datos, estos son distribuidos en las máquinas del cluster, donde las consultas son ejecutadas en alguna de las particiones, realizándose de forma paralela. Esto significa que cuando se realiza un pedido a un nodo del cluster, éste es el encargado de confirmar la respuesta, le hace llegar el pedido de manera simultánea a los restantes nodos, los cuales le devuelven la respuesta correcta que será recibida por el cliente. Esto da la idea de que el funcionamiento es basado en un nodo principal y varios nodos esclavos.

- **Control de Concurrencia**

En Plproxy se implementa el control de concurrencia de tipo multiversión, que funciona con grandes cantidades de datos permitiendo el acceso de muchos usuarios a la vez en el sistema. Su comportamiento es similar al de PGCluster, explicado anteriormente dentro de los sistemas basados en replicación, comprometiendo la transacción actual y dando comienzo a una nueva. Esta estrategia es factible pues elimina el uso de bloqueos.

2.5.2 MySQL-Cluster

MySQL Cluster es una tecnología que permite el agrupamiento de bases de datos en memoria en un entorno de no compartición. Esta arquitectura permite que el sistema funcione con *hardware* económico y sin ningún requerimiento especial de software. Tampoco tienen ningún punto de fallo porque cada componente tiene su propia memoria y disco, permitiendo también la división de bases de datos en memoria en un entorno paralelo.

Esta herramienta esta integrada, según puede observarse en la Figura 2.8, por un servidor MySQL estándar con un motor de almacenamiento clusterizado en memoria llamado NDB. Un cluster MySQL, está formado por un conjunto de máquinas, cada una ejecutando un número de procesos incluyendo servidores MySQL, nodos de datos para *NDB Cluster*, servidores de administración y posiblemente programas especializados de acceso a datos.

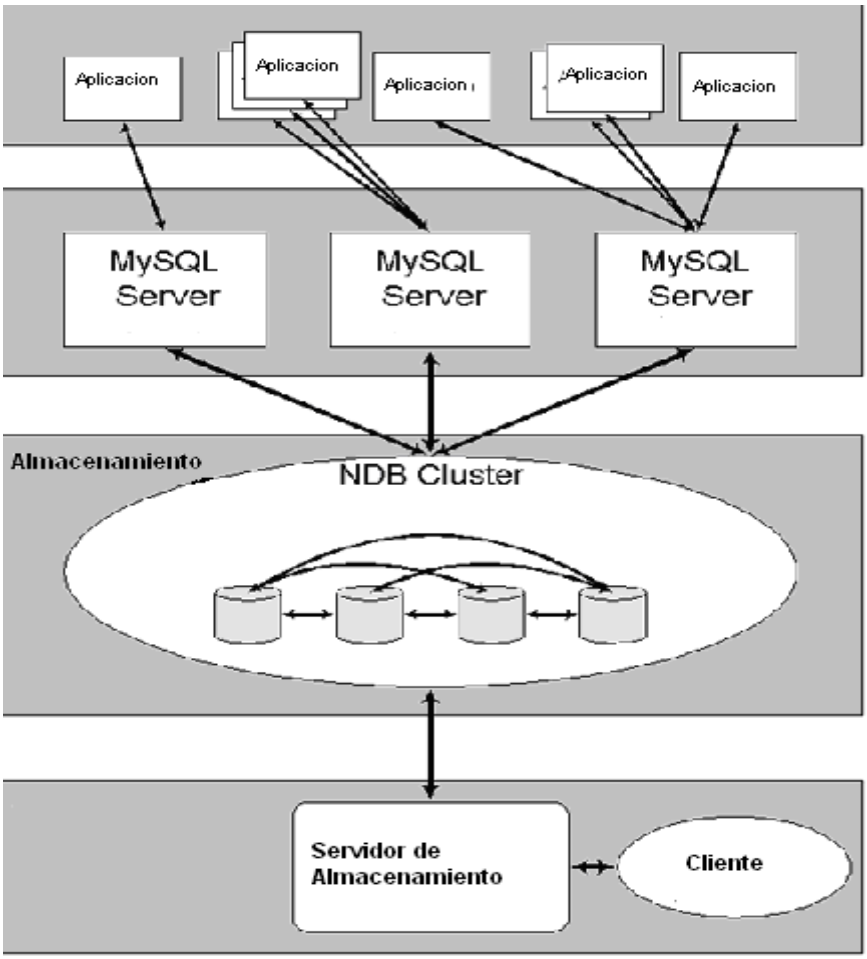

Figura 2.8 Relación de los componentes de *MySQL Cluster*

Todos estos programas funcionan juntos para formar un *MySQL Cluster*. Cuando se almacenan los datos en el motor *NDB Cluster*, las tablas se almacenan en los nodos de datos. Tales tablas son directamente accesibles desde todos los otros servidores MySQL en el cluster. Por lo tanto, en una aplicación de pago que almacene datos en un cluster, si una aplicación actualiza el salario de un empleado, todos los otros servidores MySQL que acceden a estos datos pueden ver el cambio inmediatamente.

El nodo de administración tiene como objetivo administrar los restantes nodos dentro del MySQL Cluster, y de esta forma, proporcionar datos de configuración, iniciar y detener procesos, y ejecutar copias de seguridad. El nodo SQL es el que accede a los datos del cluster. En el caso de *MySQL Cluster*, un nodo cliente es un servidor MySQL tradicional que usa el motor *NDB Cluster*. El nodo de datos, es el que almacena los datos del cluster. Hay tantos nodos de datos como réplicas, multiplicado por el número de fragmentos. Por ejemplo, con dos réplicas, cada uno teniendo dos fragmentos, necesita cuatro nodos de datos, como se muestra en la Figura 2.9

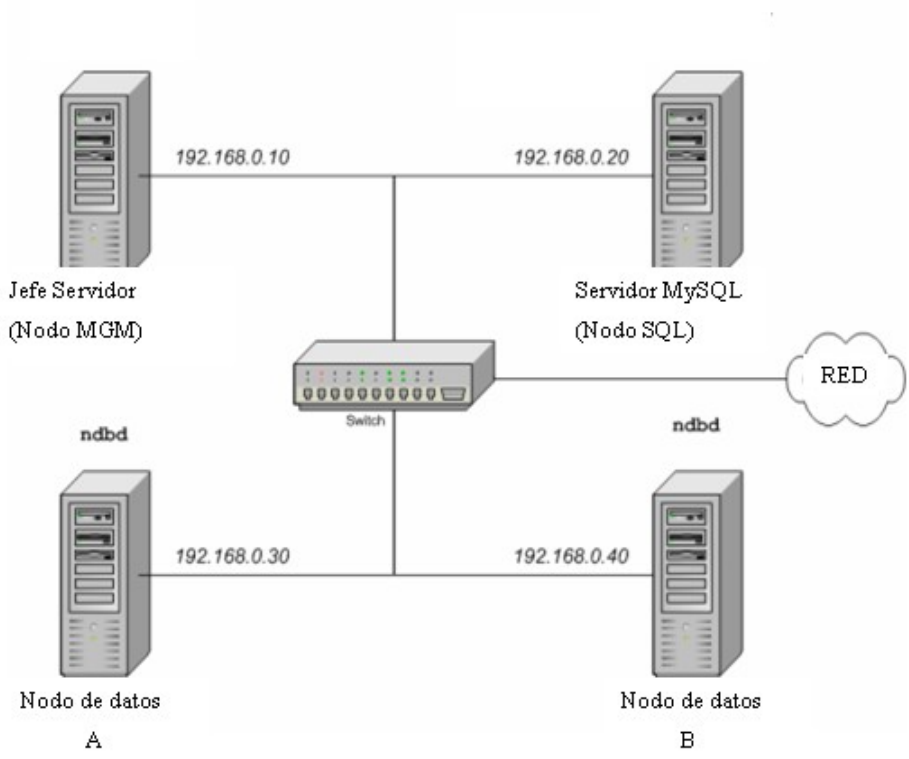

Figura 2.9 Estructura de MySQL Cluster

La configuración de un cluster implica configurar cada nodo individual en el cluster e inicializar los enlaces de comunicación individual entre los nodos. *MySQL Cluster* está diseñado con la intención de que los nodos de almacenamiento sean homogéneos en términos de procesador, espacio de memoria y ancho de banda. Además, para proporcionar un punto único de sincronización, todos los datos de la configuración del cluster se guardan en un único fichero.

El servidor de administración (nodo MGM) maneja el fichero de configuración del cluster y el *log*. El *log* es un registro secuencial que dice cómo y cuándo se han almacenado los datos en la base de datos. Cada nodo en el cluster recibe la información de configuración del servidor de administración, y necesita una forma de determinar dónde reside. Cuando ocurren eventos interesantes en los nodos de datos, se transfiere la información acerca de estos eventos al servidor de administración, que guarda la información en el *log* del cluster.

Además, puede existir cualquier número de procesos clientes del cluster o aplicaciones. Existen dos tipos de clientes: Clientes MySQL estándar y Clientes de administración. Los clientes MySQL estándar no son diferentes para *MySQL Cluster*. En otras palabras, *MySQL Cluster* puede ser accedido para aplicaciones MySQL existentes escritas en PHP, Perl, C, C++, Java, Python, Ruby, entre otros. Los clientes de administración son los que se conectan al servidor de administración y proporcionan comandos para arrancar y parar nodos, rastreo de mensajes (sólo en versiones de depuración), mostrar versiones y estado de los nodos e iniciar y detener copias de seguridad.

- **Control de concurrencia**

El modo de usuario único permite al administrador de la base de datos restringir el acceso al sistema de base de datos a un único servidor MySQL (nodo SQL). Cuando el sistema entra en el modo de usuario único, todas las conexiones a los demás servidores MySQL se cierran y todas las transacciones en ejecución se abortan, no permitiendo nuevas transacciones. Una vez que el cluster ha entrado en modo de usuario único, sólo el nodo SQL designado tiene acceso a la base de datos.

- **Protocolo de Recuperación**

Los datos almacenados en los nodos de datos de *MySQL Cluster* pueden replicarse, debido que el cluster puede tratar fallos de nodos de datos individuales sin otro impacto, además de abortar unas pocas transacciones a causa de la pérdida del estado de transacción. Como las aplicaciones transaccionales se suponen que tratan fallos transaccionales, esto no debería ser un problema. Al llevar *MySQL Cluster* al mundo *Open Source*, MySQL proporciona tratamiento de datos clusterizado con alta disponibilidad, alto rendimiento y escalabilidad, disponible para aplicaciones que manipulen grandes cantidades de información.

Existen varias formas de tratar un fallo de un nodo al ejecutar el modo de usuario único. Una de las más utilizadas es la conclusión de todas las transacciones de modo de usuario único, una vez terminada esta fase es ejecutado el comando *exit single user mode* y luego se realiza un

reinicio de los nodos del cluster. De esta forma se brinda un servicio de alta disponibilidad, escalable y eficiente para realizar la distribución cuando la dimensión de los datos es considerable.

2.6 Características fundamentales de los sistemas seleccionados

Teniendo en cuenta las funcionalidades en cada uno de los sistemas explicados en los epígrafes anteriores y la necesidad de procesamiento de grandes volúmenes de datos utilizados en la visualización científica, se decide como parte de esta investigación utilizar un grupo de herramientas que garanticen el funcionamiento deseado.

Para el trabajo posterior en esta temática se ha pretendido el uso de dos sistemas: Plproxy (sistema de distribución) y Slony-I (sistema de replicación) destacándose las mejores características determinadas de los epígrafes anteriores. Plproxy será implementado en el nodo principal con el objetivo de distribuir los datos y controlarlos de manera eficiente y Slony-I se implementará en sólo una parte de los nodos del cluster para asegurar la recuperación en caso de fallo.

Las características que siguen, fundamentan esta elección:

Plproxy

- Las herramientas contenidas por este lenguaje, permiten la distribución eficiente, posibilitándole a los usuarios dividir una base de datos entre distintos servidores.
- Permite la distribución equitativa de los datos facilitando el acceso a los mismos de forma ininterrumpible.
- Es el encargado de hacerle llegar el pedido de manera simultánea a los restantes nodos del cluster. Dicha consulta se ejecuta de forma paralela en todos los nodos, y es devuelta correctamente al cliente.
- Posibilita el acceso múltiple a las bases de datos, lográndose mediante control de

concurrencia multiversión.

- Elimina el uso de bloqueos.
- Al ser una herramienta de PostgreSQL permite la implementación de objetos.

Slony-I

- Permite replicar los datos de los nodos para caso de fallo.
- Posibilita la implementación de un sistema que preste servicios ininterrumpidamente aunque exista algún fallo en los nodos del sistema.
- Posibilita tener múltiples sistemas esclavos remotos.
- Al ser una herramienta de PostgreSQL es totalmente compatible con Plproxy.
- No contiene la detección de fallo de manera automática, pero a pesar de esto cuando un nodo falla, es reemplazado y separado para la posterior recuperación e incorporación a la base de datos.
- Permite que el usuario pueda acceder a los datos de manera efectiva, es decir ver realmente lo que le corresponde, lográndolo mediante un control de concurrencia multiversión.
- Es uno de los sistemas de replicación más recomendados para la replicación de datos.

2.7 Conclusiones Parciales

El estudio de los sistemas que contienen herramientas utilizadas para implementar bases de datos paralelas, tomando en consideración las funcionalidades que más se destacan en Oracle

RAC como software propietario, ha permitido determinar dos de las tecnologías más efectivas dentro de las que emergen en el área del software libre. Esta investigación se ha realizado con el propósito de implementar dichos sistemas para realizar un proceso eficiente de almacenamiento y gestión de grandes volúmenes de datos. Con lo anterior se pretenden obtener resultados prácticos exitosos.

Capítulo 3Implementación de las herramientas para el trabajo con bases de datos paralelas y evaluación de la eficiencia.

3.1 Introducción

En esta investigación se han tratado de forma general las tecnologías existentes que brindan soporte para la implementación de bases de datos científicas, específicamente en ambientes paralelos. Se realizó una comparación entre dichas tecnologías, resultando las más eficientes Plproxy y Slony-I. Con la aplicación de estas herramientas, el sistema que se implemente, cuenta con todas las potencialidades de PostgreSQL, considerado uno de los gestores de bases de datos más consolidados en el área de Software libre.

En este capítulo se muestra la aplicación de estas tecnologías en un caso de estudio con datos reales obtenidos del programa *Pathfinder*, concebido por la NASA. Los datos de este programa son almacenados en ficheros de tipo HDF, considerablemente grandes y con una estructura compleja por lo que se ofrece una alternativa para el almacenamiento de estos datos en una base de datos científica, implementada sobre PostgreSQL, utilizando Plproxy para realizar la distribución de los datos y Slony-I para la replicación de los mismos. Los datos han sido importados con el uso de una herramienta automática, que permite transformar los *datasets* almacenados en un fichero HDF, hacia la base de datos. Las características del cluster sobre el que se implementa el sistema, la herramienta de importación a la base de datos con la información obtenida de NASA y la estructura de la base de datos implementada sobre el cluster son temas que se abordan con profundidad en este capítulo.

3.2 Diseño de la arquitectura del Sistema de Bases de Datos Paralelas

A continuación se presenta una propuesta para la arquitectura de un sistema de bases de datos paralelas implementado sobre un cluster de computadoras donde se muestra la interacción de

este sistema con la herramienta HDF2PDB, que permite importar datos desde el fichero HDF, utilizando las bibliotecas necesarias y ejecutando funciones implementadas en el SGBD PostgreSQL. Mediante esta herramienta los datos pueden ser insertados automáticamente en los nodos del cluster de forma distribuida, brindándole al sistema disponibilidad y balanceo de carga. Las características de la herramienta son explicadas con posterioridad.

Detallando en la arquitectura del sistema de bases de datos paralelas, en primer lugar es necesario definir las características del cluster sobre el que funciona, en este caso es un cluster de la categoría *shared-nothing* formado por 10 computadoras conectadas a un *switch,* donde son utilizadas 9 de ellas, como aparece en la Figura 3.1

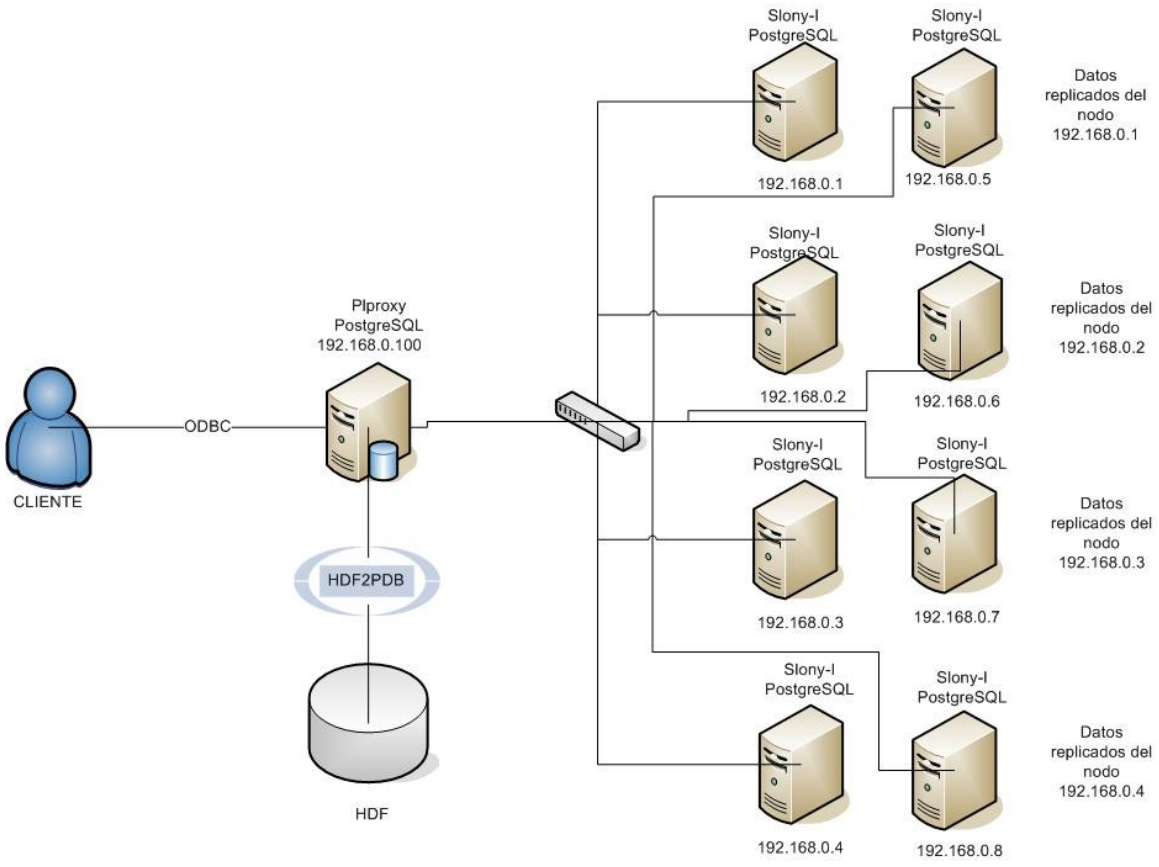

Figura 3.1 Arquitectura del sistema de base de datos paralela

Una de las computadoras actúa como servidor y el resto como nodos esclavos. De ellos, 4 contienen los datos y los restantes, los datos replicados, logrando así la recuperación en caso de pérdida de la información.

Esta investigación se ha centrado principalmente en implementar un sistema que sea capaz de gestionar la información de forma eficiente, aprovechando la capacidad de cómputo y de almacenamiento de un sistema paralelo. Los datos que representan el caso de estudio, son utilizados en la visualización, por lo que las aplicaciones clientes deben ser capaces de conectarse a la base de datos en PostgreSQL con algunos de los protocolos soportados por este gestor. Estas aplicaciones no pueden acceder directamente a las tablas de la base de datos, pues estos objetos se encuentran distribuidos entre los nodos del cluster. El acceso se realiza a través de funciones previamente implementadas en el gestor, que son las encargadas de gestionar los datos en el cluster y brindar una respuesta coherente a las aplicaciones cliente. Es importante señalar que no es posible ejecutar las tradicionales consultas en SQL sobre el Proxy, pues este sólo es el encargado de redireccionar dichas consultas hacia los nodos donde se encuentra la información.

En este caso de estudio se han combinado los sistemas de Plproxy y Slony-I para lograr el funcionamiento deseado. Como se menciona en el capítulo anterior, Plproxy es un lenguaje procedural para ejecutar llamadas a bases de datos remotas, útil para implementar el proceso de distribución en un sistema paralelo. Slony-I es una de las herramientas más elaboradas en la categoría de software libre, que permite implementar la replicación entre los nodos de un cluster, brindando escalabilidad al sistema. En estas herramientas, diseñadas para ejecutarse sobre SGBD PostgreSQL se profundizará a continuación, pero antes es necesario conocer la estructura de los datos objeto de estudio.

3.3 Análisis de la estructura de los Datos Científicos

Para la evaluación de las herramientas implementadas, los datos utilizados como caso de estudio fueron obtenidos del programa *Pathfinder*, concebido por la NASA a principios de los años 90, con el objetivo de poner a disposición de la comunidad científica los datos tomados durante años por varios satélites que recopilan y procesan a diario información atmosférica. Uno de los componentes de este programa es TOVS (TIROS[1] Operational Vertical Sounder), que genera diferentes conjuntos de datos a partir de la información almacenada por *Pathfinder*. Estos datos incluyen mediciones desde julio de 1979 hasta la actualidad, y tienen una resolución espacial de 100 km.

Los conjuntos de datos generados por TOVS se encuentran almacenados en archivos de tipo HDF, que son procesados utilizando complejos algoritmos para obtener información climatológica de las áreas correspondientes.

De manera general los archivos HDF representan un formato de datos multi-objetos para la transferencia de datos gráficos y numéricos entre computadoras. Cada modelo es representado de forma diferente, debido a que pueden definirse nuevos tipos, y es necesario implementar aplicaciones para su lectura, escritura y organización. Entre los modelos que soporta HDF se encuentran los arreglos multidimensionales, imágenes y tablas.

Los archivos de tipo HDF son autodescriptivos, pues permiten a las aplicaciones conocer información acerca de su estructura interna, con el objetivo de realizar análisis de los datos sin ninguna información adicional. Son flexibles, extensibles y portables. Brindan a los usuarios un grupo de bibliotecas para su empleo, por lo que la implementación de un sistema que funcione de forma similar resulta una tarea difícil.

El conjunto de datos utilizado en este trabajo, TOVS5DAY.h5, almacena información referente al clima de la región durante un período de 5 días. De este conjunto se seleccionaron para este

[1] TIROS son las siglas en inglés de Television Infrared Observation Satellite.

estudio las variables temperatura, presión y nivel de humedad, almacenadas como arreglos tridimensionales de 5x180x360 de números reales.

3.4 Configuración e implementación de ScientificPDB

En los epígrafes que continúan se hará referencia a los elementos que componen ScientificPDB a partir de la modelación de los datos científicos de un caso de estudio. Se explica la instalación y configuración de Plproxy y Slony-I así como la forma en que se implementan las funciones en los nodos regulares y en el nodo Proxy.

3.4.1 Modelación de los Datos Científicos del caso de estudio

Cada una de las variables consideradas del conjunto de datos de TOVS *Pathfinder* está definida como un arreglo tridimensional. La forma estándar de representar estructuras en una base de datos relacional es usar el esquema estrella o copo de nieve en el cual cada dimensión se descompone en su propia tabla y se enlaza con la variable dependiente mediante una relación de llave extranjera. En un SGBD Objeto Relacional la variable dependiente puede ser almacenada junto con sus dimensiones usando una columna que contenga un arreglo multidimensional, implementado como un tipo de datos abstracto. Esta representación resulta más natural, ya que la variable dependiente se almacena en la misma fila que las variables independientes o dimensiones.

PostgreSQL es un SGBD Objeto Relacional que permite la definición de tipos abstractos y las funciones que trabajan sobre ellos. Además, tiene un tipo predefinido de arreglo n-dimensional dinámico que se puede utilizar para modelar datos científicos. Este modelo tiene la ventaja de que se pueden insertar, actualizar y eliminar elementos individuales, así como definir índices sobre ellos, posibilidad que no aparece sobre tipos de datos similares de otros SGBD Objeto Relacional, como Oracle 10g.

```
CREATE TYPE PressureData AS (
      description text,
      pdata real[5][180][360]
);
CREATE TABLE PressureTable (
      firstDay date,
      data PressureData
);
```

Figura 3.2 Modelado de la variable presión PostgreSQL

En la Figura 3.2 se muestra la forma en que ha sido modelada la variable presión en PostgreSQL. Nótese que se creó un tipo de datos abstracto (*PressureData*) que encapsula los valores numéricos de la presión y una descripción de los mismos. De manera similar se pueden agregar otros atributos. El resto de las variables fueron modeladas análogamente.

Las aplicaciones científicas pueden generar grandes cantidades de datos del orden de los terabytes en cada ejecución. Esta información se almacena generalmente en alguno de los formatos de datos científicos mencionados anteriormente. Para poblar una base de datos es necesario extraer la información de los archivos donde están almacenados. En el caso de estudio que aquí se presenta los datos están almacenados en archivos HDF.

El proceso de extracción e inserción en la base de datos puede ser engorroso y consumir tiempo, pues no están disponibles herramientas que lo realicen de manera automática. Para completar esta tarea, en este trabajo, se creó una herramienta en lenguaje C (HDF2PDB), que recibe como parámetros el nombre del archivo HDF y la información de la base de datos (nombre, nombre de la tabla, nombre de usuario y contraseña, puerto, etc.), e inserta en la base de datos la información importada del archivo HDF. Para ello utiliza las bibliotecas libhdf5 y libpq, soportadas por los paquetes HDF y PostgreSQL, respectivamente (ver epígrafe 3.4).

Se utilizó además el paquete ECPG, que ofrece una biblioteca de funciones y un preprocesador para interactuar con bases de datos desde el lenguaje C. Con el uso de este paquete es posible incluir llamadas a funciones implementadas en lenguajes procedurales desde el código fuente, lo que facilita la realización de tareas comunes con bases de datos.

3.4.2 Instalación y configuración de los sistemas seleccionados

Plproxy es un lenguaje procedural incorporado a PostgreSQL para realizar llamadas remotas a bases de datos. Con este lenguaje es posible implementar un sistema de bases de datos que funcione de forma paralela, a través de la distribución de los datos entre los nodos del cluster.

Para garantizar que esta información esté disponible en todo momento se ha implementado el proceso de replicación utilizando otra de las herramientas de PostgreSQL, Slony-I. En esta sección se explica cómo utilizar este lenguaje para implementar consultas a las bases de datos remotas que se encuentran distribuidas en el cluster y cómo replicar los datos en los nodos del cluster.

3.4.2.1 Instalación y configuración de Plproxy

Para realizar la instalación de Plproxy es necesario tener instalado un ambiente de desarrollo de PostgreSQL. Posteriormente deben ejecutarse los siguientes pasos:
Descargar Plproxy desde http://pgfoundry.org/projects/Plproxy.
Construir Plproxy ejecutando *make* y *make install* dentro del directorio de Plproxy. Si se presentan problemas en la instalación asegurarse de que el archivo de configuración pg_config está en los directorios del PATH
Para instalar Plproxy en una base de datos específica, ejecute el fichero Plproxy.sql que se encuentra en los programas fuentes. Por ejemplo ejecutando:
`psql -f $SHAREDIR/contrib/Plproxy.sql mydatabase`

Estos pasos pueden ser evitados si la instalación se realiza desde un paquete.

Utilizar esta herramienta para distribuir los datos entre los nodos del cluster requiere que se implementen algunas funciones de configuración. Cuando una consulta necesita ser reenviada hacia una base de datos remota se ejecuta la función plproxy.get_cluster_partitions(cluster) para obtener la información necesaria para establecer la conexión:

```
CREATE OR REPLACE FUNCTION plproxy.get_cluster_partitions(cluster_name text)
 RETURNS SETOF text AS $$
 BEGIN
    IF cluster_name = 'usercluster' THEN
       RETURN NEXT 'dbname=part00 host=192.168.0.1';
       RETURN NEXT 'dbname=part01 host=192.168.0.2';
       RETURN NEXT 'dbname=part02 host=192.168.0.3';
       RETURN NEXT 'dbname=part03 host=192.168.0.4';
       RETURN;
    END IF;
    RAISE EXCEPTION 'Unknown cluster';
 END;
 $$ LANGUAGE plpgsql;
```

Figura 3.3 Función de configuración de Plproxy

La configuración del cluster puede ser almacenada en alguna tabla en la base de datos del Proxy y la cantidad de nodos tiene que ser una potencia de dos.

Para asegurarse que las consultas que se están realizando deben ejecutarse sobre un cluster determinado, es necesario definir una función que verifique esto y retorne un mensaje de error si se quieren ejecutar consultas sobre un cluster que no existe. Ver figura 3.4.

```
CREATE OR REPLACE FUNCTION
plproxy.get_cluster_version(cluster_name text)
  RETURNS int4 AS $$
  BEGIN
     IF cluster_name = 'usercluster' THEN
        RETURN 1;
     END IF;
     RAISE EXCEPTION 'Unknown cluster';
  END;
  $$ LANGUAGE plpgsql;
```

Figura 3.4 Función de versiones de Plproxy

Por último es necesario implementar una función que se encargue de proveer los valores para la configuración del cluster, esta función se define como:

```
CREATE OR REPLACE FUNCTION plproxy.get_cluster_config(
    in cluster_name text,
    out key text,
    out val text)
  RETURNS SETOF record AS $$
  BEGIN
     -- lets use same config for all clusters
     key := 'connection_lifetime';
     val := 30*60; -- 30m
     RETURN NEXT;
     RETURN;
  END;
  $$ LANGUAGE plpgsql;
```

Figura 3.5 Función de particiones de Plproxy

Esta función retorna pares de la forma (llave, valor), donde las llaves pueden ser:

- connection_lifetime: Tiempo de vida de la conexión. Plproxy eliminará configuraciones anteriores.
- connect_timeout: Cancela la conexión inicial si el tiempo supera al de esta configuración, por lo que hay que duplicar la conexión.

- query_timeout: Si el resultado de la consulta no se ha recibido en este tiempo, se cierra la conexión.
- disable_binary: Permite deshabilitar la transferencia de datos binarios entre los nodos del cluster.

Estas funciones permiten que el sistema se encuentre listo para implementar la distribución de los datos entre los nodos del cluster, que serán explicadas en el próximo epígrafe.

3.4.2.2 Instalación y configuración de Slony-I

La instalación de Slony-I es un poco más simple. Al igual que en Plproxy, se deben obtener los fuentes en el mismo sitio. Después es necesario descompactarlos de la siguiente manera:

```
gunzip slony.tar.gz;
tar xf slony.tar
```

Se creará entonces un directorio que será el centro del resto del proceso de instalación. El primer paso en este proceso es configurar el sistema de archivos fuentes. Esto se realiza usualmente ejecutando un *script* de configuración. Slony-I necesita conocer la dirección de algunas bibliotecas, archivos binarios y elementos de la instalación de PostgreSQL para su correcto funcionamiento. Para detallar mejor estas opciones se puede ejecutar el comando de la figura 3. 6

```
PGMAIN=/opt/dbs/pgsql \
./configure \
   --prefix=$PGMAIN \
   --bindir=$PGMAIN/bin \
   --datadir=$PGMAIN/share \
   --libdir=$PGMAIN/lib \
   --with-pgconfigdir=$PGMAIN/bin \
   --with-pgbindir=$PGMAIN/bin \
   --with-pgincludedir=$PGMAIN/include \
   --with-pglibdir=$PGMAIN/lib \
   --with-pgsharedir=$PGMAIN/share
```

Figura 3.6 Instalación de Slony-I

Para comenzar el proceso de construcción de los fuentes, si está utilizando Linux GNU, ejecute el comando *make all*. Si el proceso termina con éxito la última línea debe ser:

```
All of Slony-I is successfully made. Ready to install.
```

Figura 3.7 Mensaje de éxito de la instalación de Slony

Para instalar, ejecute el comando *make istall*. Este proceso instalará archivos en el directorio de instalación de PostgreSQL por lo que debe asegurarse de tener permiso de escritura en este directorio. Si todos estos pasos se ejecutan de forma exitosa el sistema se encuentra listo para pasar el próximo paso, la configuración.

Antes de comenzar a configurar esta instalación de Slony-I, se hace necesario conocer algunos conceptos que se tratarán en este proceso y que tienen una particular importancia.
Por ejemplo, el concepto de cluster ha sido tratado ampliamente en este trabajo, sin embargo para Slony-I, un cluster es simplemente un conjunto de bases de datos entre las cuales se realiza el proceso de replicación. Así, para cada base de datos que pertenece a un cluster determinado se creará un *namespace* con el nombre del cluster.

El otro concepto importante a tratar es el de nodo de Slony-I. Un nodo es una de las bases de datos del cluster y está definida en cada *script* de Slony-I como:

```
NODE i ADMIN CONNINFO = 'dbname=testdb host=server1 user=slony';
```

Figura 3.8 Configuración de un nodo de Slony-I

De esta manera quedan definidos los dos elementos básicos de Slony-I: un cluster y un conjunto de nodos con un *namespace* que identifique a que cluster pertenecen. De esta forma, se puede definir un conjunto de replicación entre los nodos de un cluster Slony-I, como el flujo de datos que necesita ser replicado dentro de todo el sistema.
En un mismo cluster pueden existir varios conjuntos de replicación, que usualmente son diferentes pues el origen de los datos llamado nodo *master* es diferente. Este nodo es el único lugar donde las aplicaciones de usuarios pueden modificar los datos del cluster, pues representan el origen de la replicación. El resto de los nodos están configurados para recibir datos, en forma de cascada, desde el origen en el proceso de replicación.

En cada nodo corre un proceso llamado *slon* que es el encargado de manejar la replicación en dicho nodo. Entre las actividades más importantes que se realizan están: manejar los eventos de configuración y los eventos de sincronización.

En especial, la configuración se realiza en el momento en que se ejecuta el script de Slony-I, llamado *Slonyk* donde se crean las tablas de configuración, los procedimientos almacenados y los *triggers* que necesita el sistema para su funcionamiento. En el Anexo 1 se muestra el *script* utilizado en este caso de estudio.

Después que han sido configuradas las bases de datos en cada uno de los nodos que participan en este proceso de replicación, es necesario ejecutar los *slons* de cada nodo. Ver figura 3.9.

```
slon $CLUSTER "dbname=$DBNAME1 user=$SLONY_USER"
```

Figura 3.9 Ejecución del proceso slon

Esta operación debe ser ejecutada en cada nodo del cluster. En este momento se intercambia gran cantidad de información con el objetivo de garantizar la sincronización, pero todavía no se han comenzado a replicar los datos. Para comenzar la replicación es necesario ejecutar otro

script donde se informe al sistema quién representa el origen y el destino en este proceso. El *script* utilizado para comenzar el proceso de replicación en este caso de estudio se encuentra en el Anexo 2.

En el momento en que termina de ejecutarse el último *script* de configuración, los nodos esclavos comienzan a copiar datos desde los nodos orígenes. Este proceso se repite copiando cada cierto tiempo la información almacenada en los *logs* de replicación, manteniendo actualizada las copias de las bases de datos que se encuentran en los nodos esclavos.

3.4.3 Implementación de las funciones en el nodo Proxy y en los nodos regulares

Antes de realizar el proceso de importación de los datos desde el fichero HDF es necesario definir algunas funciones en el SGBD PostgreSQL, que sirvan como base para insertar los datos en las particiones indicadas. Este proceso se basa en la implementación de prototipos de funciones en el proxy, y el cuerpo de las funciones se implementa en cada nodo, de forma tal que el proxy sólo se encargue de enviar las consultas a los nodos donde se van a ejecutar. En ocasiones estas consultas pueden resultar tan simples que sólo es necesario programarlas en el proxy, especificando la sentencia en SQL y el nodo donde debe ejecutarse. Para esta investigación se han definido dos tipos de funciones: las funciones de inserción y las funciones implementadas para realizar consultas con el objetivo de realizar comparaciones de eficiencia.

Como la información científica es generalmente histórica, los datos ocasionalmente cambian. A partir de estas características, solamente se analizarán consultas de inserción, pues raramente serán necesarias consultas de actualización.

Es necesario definir un prototipo de consulta en el Proxy, en lenguaje Plproxy. En cada nodo, donde no se ha instalado Plproxy las consultas se implementarán en PlpgSQL, una extensión de SQL para el SGBD PostgreSQL, que brinda importantes potencialidades al lenguaje. Aunque no es un objetivo de esta investigación profundizar en PlpgSQL, es importante señalar que con

el uso de esta herramienta es posible implementar consultas más flexibles en SQL, la inclusión de nuevos tipos de datos y el uso de una variedad de estructuras de control.

Como en cada nodo existe una copia de la tabla de datos y como la variable en estudio presión, fue recuperada en forma de intervalos, específicamente valores menores a 300 barímetros, entre 300 y 600, entre 800 y 1200 y mayores a 1200, la función de fragmentación es muy sencilla ya que cada rango de datos es insertado en cada nodo del cluster. No es necesario definir una función de inserción en cada nodo, solamente es necesario ejecutar el código en el Proxy e indicarle en qué nodo debe ejecutarse. De esta forma, un prototipo de función de inserción es el siguiente:

```
CREATE OR REPLACE FUNCTION InsertScientificData (SData ARRAY [] [], partition int)
        Return text as $$
             CLUSTER 'usercluster';
              RUN ON partition;
              INSERT INTO tablename VALUES ("presión", "medida", SData)
               ...
        $$ Lenguaje plproxy;
```

Figura 3.10 Función de inserción

Esta función recibe como parámetros un arreglo con la información del *dataset* y un número entero que especifica la partición donde deben ser insertados estos datos. Es importante señalar que con las ventajas del uso de PostgreSQL pueden ser insertados tanto los datos en forma numérica como la información acerca de estos datos, los metadatos. Existen varios enfoques para almacenar los metadatos, pues resulta una estructura muy útil para obtener información acerca de los datos sin necesidad de analizar otras fuentes. En este trabajo se decidió, debido a la flexibilidad de PostgreSQL, almacenar los metadatos junto a los datos numéricos.

Es objetivo de esta investigación realizar pruebas de eficiencia, a través de la implementación de consultas de lectura a la base de datos. Para ello se han definido tres clases de consultas de

acuerdo a la información que se puede recuperar de la variable presión. En primer lugar es necesario definir consultas que sean capaces de recuperar los datos de presión en un área determinada, estos datos no se encuentran en un nodo específico por lo que la consulta debe ejecutarse en todos los nodos. El prototipo de función para este caso es el siguiente:

```
CREATE OR REPLACE FUNCTION getPresionbyPlace(int lamin, lamax, lomin, lomax)
RETURN Set Of  $$
   CLUSTER 'usercluster';
   RUN ON ALL;
$$ LENGUAJE plproxy
```

Figura 3.11 Función de selección en proxy

Esta función realiza una llamada dada en valores de longitud y latitud mínimas y máximas a las instancias implementadas en los nodos con las dimensiones de un área específica. La función en cada nodo, es la encargada de retornar los valores de presión que se encuentran disponibles y retornarlos al proxy, quien se encarga de formar una respuesta coherente de los datos obtenidos y se los envía a la aplicación.

De esta misma forma se implementaron consultas que retornen los lugares que exceden un valor determinado de presión, con el objetivo de ubicar áreas de altas y bajas presiones.

```
CREATE OR REPLACE FUNCTION getPresionbyPlace (int lamin, lamax, lomin, lomax)
RETURN Set Of  $$
  SELECT pressuretable.data.pdata[1][lamin:lamax][lomin:lomax]
  FROM    pressuretable
$$ LENGUAJE PlpgSQL
```

Figura 3.12 Función de selección en nodos regulares

Son destacables las facilidades para el trabajo con arreglos que brinda PostgreSQL permitiendo acceder a rangos de índices dentro de esta estructura. En este caso se ha implementado la función para el primer nodo del cluster. Las funciones análogas en el resto de los nodos tendrían el número del nodo como primer índice del arreglo. A primera vista esto pudiera resultar innecesario (tener un índice que nunca varía su valor, o sea que en el nodo uno siempre su valor va a ser uno), pero esto ocurre debido a que existen tantos intervalos de presión como nodos tiene el cluster, en este caso de estudio. Si se aumenta el número de intervalos de presión para ser más específicos, es necesario programar una función de asignación que decida cuáles intervalos deben ser ubicados en cierto nodo y de esta forma se tendrían que diferenciar estos valores por el primer índice del arreglo.

Se realizaron además, consultas combinadas que buscan valores extremos de presiones en áreas determinadas.

De esta forma quedan definidas las funciones para ser utilizadas, tanto por la herramienta de importación para poblar la base de datos como las que van a ser ejecutadas por las aplicaciones para realizar pruebas de eficiencia.

3.5 Implementación de HDF2PD

Cuando la estructura de la base de datos paralela se encuentra lista, es necesaria una herramienta que sea capaz, de forma automática, de acceder a los datos en el archivo HDF, para esto fue implementada HDF2PDB.

El lenguaje seleccionado para esta implementación fue C, debido principalmente al soporte que brinda para el trabajo con HDF y con PostgreSQL. Con la biblioteca libhdf.lib, que propone este lenguaje para gestionar archivos HDF se confeccionó la primera parte de la herramienta, básicamente el acceso al archivo y la selección del *dataset* a importar.

Es importante señalar que para acceder directamente a un *dataset* desde un lenguaje de programación es necesario conocer el nombre del mismo. Sin embargo, si se quiere acceder secuencialmente a todos los objetos, existen funciones dentro de esta biblioteca que permiten hacerlo.

En esta investigación no se realizó una búsqueda secuencial debido a que sólo se importaron algunas variables almacenadas en el archivo. Aunque es posible importar los atributos de un *dataset*, sobre todo aquellos que brindan información acerca de los datos y al archivo HDF. Respecto a las variables de este caso de estudio resultó importante conservar las unidades de medida.

Después de importados, los datos se encuentran temporalmente en un arreglo tridimensional, para ser utilizados por la biblioteca libpq que permite ejecutar consultas desde aplicaciones clientes escritas en C a una base de datos en PostgreSQL. Esta biblioteca contiene un grupo de funciones que pueden ser accedidas también por otros lenguajes como Python, C++, tcl, Perl y ECGP. Para utilizar libpq, los programas deben incluir en su cabecera el archivo libpq-fe.h y asegurarse de estar enlazado con la biblioteca.

Con el uso de esta herramienta resulta relativamente sencillo ejecutar llamadas a las funciones previamente implementadas en la base de datos, con el objetivo de poblar la misma con los datos del arreglo. Igualmente son insertados los atributos del *dataset.*

El proceso de distribución entre los nodos es casi transparente para la aplicación, por lo que esta herramienta puede ser utilizada indistintamente para importar datos hacia una base de datos centralizada o hacia el prototipo de base de datos paralela que se presenta en esta investigación.

Segmentos del código de dicha herramienta aparecen en el Anexo 3 de este documento. Al concluir este paso, la base de datos está lista para ser consultada y realizar pruebas con el objetivo de medir la eficiencia de las herramientas propuestas en este trabajo. Los archivos

HDF con los que trabaja la herramienta implementada pueden ser, como se analizó en el capitulo uno, tan complejos como sea el área de investigación en que se emplean, es por eso que una de las recomendaciones de esta investigación es ampliar esta herramienta para que sea capaz de analizar completamente un fichero HDF, de forma recursiva, por la existencia en la mayoría de los casos de grupos de *datasets* y modelar una base de datos para recibir la información.

3.6 Comparación entre la solución propuesta y los formatos estándares

Las funciones implementadas sobre el nodo proxy y los nodos regulares analizadas con anterioridad formaron parte del proceso de evaluación de la solución propuesta en cuanto a funcionamiento. Esta evaluación se realizó en base a una comparación con el funcionamiento de los mismos datos almacenados en un fichero HDF utilizando las herramientas que brinda este formato para realizar consultas. Con el objetivo de minimizar el efecto de la información almacenada en *cache* que pudiera afectar los resultados, todas las ejecuciones de las consultas siguieron los pasos mostrados en la Tabla siguiente:

Base de Datos Paralela	Formato de Datos Científicos
1. Apagar el gestor. 2. Desmontar el sistema de archivos 3. Montar el sistema de archivos. 4. Iniciar el gestor 5. Ejecutar las consultas	1. Desmontar el sistema de archivos 2. Montar el sistema de archivos. 3. Ejecutar las consultas

Tabla 3.1 Pasos para ejecutar las consultas (Kohen and Hurley, 2006)

Las consultas se ejecutaron sobre *datasets* de pequeña y mediana escala e incluyeron el acceso, en el caso de las consultas a la base de datos paralela, a un único nodo y a varios nodos. Los resultados de las cinco ejecuciones exitosas se muestran a continuación.

	Datasets pequeño	*Datasets* mediano
	13.01s	20.15s
BD	14.80s	30.32s

Tabla 3.2 Consultas sobre un nodo

P		

	Datasets pequeño	*Datasets* mediano
	7.97s	13.92s
BDP	2.17s	13.19s

Tabla 3.3 Consultas sobre varios nodos

Como se ha mencionado los datos fueron distribuidos en los nodos de acuerdo a sus niveles de presión, almacenando en cada uno un punto del espacio de muestra y su presión correspondiente. Para realizar consultas sobre un único nodo se encuestó la base de datos respecto a su nivel de presión, debido a que las consultas son enviadas primeramente al proxy, quien determina donde se encuentran los datos en cuestión. Para realizar consultas en todos los nodos simplemente se seleccionaron datos de todos los niveles, de esta forma el proxy debe enviar las consultas a todos los nodos.

Los tiempos mostrados se obtuvieron con la herramienta *iostat*, una utilidad de Linux para monitorear las tareas de entrada salida. Este resultado sugiere que la solución propuesta no es mejor cuando las consultas son ejecutadas solamente en un nodo, sin embargo cuando la carga se distribuye entre varios nodos se obtienen mejores resultados que en el formato de datos científicos. También se debe señalar que la base de datos paralela mejora su funcionamiento en la medida en que el número de elementos por *dataset* es más pequeño.

3.7 Conclusiones Parciales.

La implementación de la herramienta HDF2PDB, ha permitido la importación eficiente de los datos desde un archivo HDF hacia una base de datos paralela. Se implementaron las tecnologías más efectivas escogidas dentro de la categoría de software libre, en un cluster de computadoras, permitiendo la distribución de los datos en los diferentes nodos del cluster y la

replicación de los mismos, así como la recuperación en caso de pérdida de información. Se realizaron consultas, las cuales se ejecutaron en forma paralela en los diferentes nodos del cluster y se realizaron pruebas que arrojaron importantes resultados que aunque no fueron definitivos muestran que el uso de bases de datos para gestionar información científica es una interesante tarea para futuras investigaciones.

Conclusiones

1. Se seleccionaron las tecnologías más apropiadas utilizadas en el almacenamiento y gestión de la información científica para su integración en un SGBD.

2. Se determinaron las herramientas de software libre disponibles para implementar un sistema de bases de datos paralelos que gestione datos científicos y entre ellas se seleccionaron las más consolidadas.

3. Se determinó la forma más apropiada de vincular los formatos nativos de datos científicos con un SGBD.

4. Se desarrolló un caso de estudio en el que se implementaron consultas de visualización científica en una base de datos implementada sobre un clúster de computadoras.

5. Se implementó una herramienta que permite importar datos almacenados en el formato HDF a un SGBD paralelo.

Recomendaciones

1. Extender la herramienta de importación de datos para el trabajo con ficheros HDF más complejos u otros formatos de datos científicos.

2. Utilizar distribuciones más actualizadas de las herramientas utilizadas con el objetivo de obtener mejores resultados.

3. Dada la variedad de posibles soluciones para la implementación de bases de datos paralelas usando software libre, el estudio realizado no agota este campo de investigación. Otras soluciones pudiesen ser estudiadas con más profundidad para su posterior implementación.

Bibliografía

ADELMANN, E. W. (2006) H5PART a Portable High Performance Parallel Data Interface for Electromagnetic Simulations.

ALLCOCK, B. (2001) High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies.

ALSBERG, B.K., KIRKHUS, L., HAGEN, R., KNUDSEN, O., TANGSTAD T. & ANDERSSEN E. (2003) Zherlock: an Open Source Data Analysis Software

ARKOR, B. (2001) Data and Metadata Collections for Scientific Applications.

BAIÃO, F.A., MATTOSO, M. Y ZAVERUCHA, G. (1998) Towards an Inductive Design of Distributed Object Oriented Databases. *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems.* New York City, NY, USA, IEEE Computer Society

BAIÃO, F.A., MATTOSO, M. Y ZAVERUCHA, G. (2004) A Distribution Design Methodology for Object DBMS. *Distributed and Parallel Databases,* vol. 16**,** num. 1, 45-90.

BOSE, R. & FREW, J(2005)Lineage Retrieval for Scientific Data Processing: A Survey. *ACM Computing Surveys*, Vol. 37, No. 1.

BRAHMADATHAN, K. Y RAMARAO, K.V.S. (1992) On the design of replicated databases. *Inf. Sci*., vol. 65, num. 1-2, 173-187.

BUNEMAN, P., KHANNA, S. & TAJIMA, K. (2004) Archiving Scientific Data. *ACM Transactions on Database Systems*, Vol. 29, No. 1.

CHAKRABARIT, S. & MUTUCRHISMAS, S. (1996) Resource scheduling for parallel database and scientific applications.

CHEN, K. & LIU, L. (2006) iVIBRATE: Interactive Visualization-Based Framework for Clustering Large Datasets. *ACM Transactions on Information Systems*, Vol. 24, No. 2.

CHUNDI, P., ROSENKRANTZ, D.J. Y RAVI, S.S. (1996) Deferred Updates and Data Placement in Distributed Databases. En: S.Y.W. Su (Ed.) *Proceedings of the Twelfth International Conference on Data Engineering.* New Orleans, Louisiana, IEEE Computer Society.

COULON, C., PACITTI, E. Y VALDURIEZ, P. (2005) Consistency Management for Partial Replication in a High Performance Database Cluster. *IEEE International Conference on Parallel and Distributed Systems (ICPADS2005).* Fukuoka, Japan.

DEWITT, D. J. & GRAY, J. (1992) Parallel Database System: The Future of High Performance Database Processing.

DUDOIT, S. (2003) Open Source Software for the Analysis of Microarray Data.

EDWARD, E. & FETROW, S. (2007) PDBSQL: A Storage Engine for Macromolecular Data. ACMSE.

FOREMAN, G. & ZANG, B. (2000) Distributed Data Clustering can be Efficient and Exact. *SIGKDD Explorations*. Volume 2, Issue 2.

GALLOP, J. (1994) Underlying Data Models and Structures for Visualization

GONG, B., SINGH, R. & JAIN, R.(2004) ResearchExplorer: Gaining Insights through Exploration in Multimedia Scientific Data.

GRAY, J. & LIU, D.(2005) Scientific Data Management in the Coming Decade.

GRAY, J.(2002) Data Mining the SDSS SkyServer Database.

GROUP, H. (2007) HDF5 A New Generation of HDF the Hierarchical Data Formats.

HADDELTON, R.(2006) Client/Server Architecture in the ADAMS Parallel, Object-Oriented Database System.

HANSEN, C. D. & JOHONSON, C. R.(Eds.)(2005) *The Visualization Handbook*. ELSEVIER

HAINING, R. (2000) Providing Scientific Visualization for Spatial Data Analysis: Criteria and an assessment of SAGE.

HEIMANS, J.(2002) An introduction to Distributed Visualization.

HELLMAN, E. (2007) Evaluation of Database Management Systems for Erlang. *ACM Transactions on Database Systems*, Vol. 56, No. 3.

KARAVANIC, K. & MAY, J. (2005) Integrating Database Technology with Comparison-based Parallel Performance Diagnosis: The PerfTrack Performance Experiment Management Tool. *Proceedings of the 2005 ACM/IEEE SC*.

KEIM, D. (2002) Information Visualization and Visual Data Mining. *IEEE Transactions on Visualization and Computer Graphics*, vol. 7, no. 1.

KEMME, B. (2000) Database Replication for Clusters of Workstations.

KIEV, K. (2007) PL/proxy tutorial.

KIEV, K. (2007) PostgreSQL cluster: partitioning with Plproxy.

KOHEN, S. & HURLEY, P. (2006) Scientific Formats for Object-Relational Database Systems: A Study of Suitability and Performance.

LI, J., LIAO, W. & CHOUDHARY, A. (2003) Parallel netCDF: A Scientific High-Performance I/O Interface.

MITANI, A. (2007) PGCluster-II: Clustering system of PostgreSQL using Shared Data.

PACCITI, E. & ÖZSU, M. (2007) Preventive Replication in a Database Cluster.

PFALTZ, J.(2000) Scalable, Parallel, Scientific Databases.

SHANKAR, S. (2007) Data Driven Workflow Planning in Cluster Management Systems.

SILBERSCHATZ, A. & KORTH, H. F. (1998) *Fundamentos de Bases de Datos*, MacGraw-Hill.

STOLTE, E. & ALONSO, G. (2002) Efficient Exploration of Large Scientific Databases. Proceedings of the 28th VLDB Conference.

TAKHUR, R. & NO, J. (2000) Integrating Parallel File I/O and Database Support for High-Performance Scientific Data Management.

WOLF, M. (2002) SmartPointers: Personalized Scientific Data Portals In Your Hand. IEEE.

Anexo 1. Slonyk 1.0

```
#!/bin/sh

CLUSTER=usercluster
DBNAME1=part00
DBNAME2=part00copy

HOST1=127.0.0.1



SLONY_USER=postgres
PGBENCH_USER=root

slonik <<_EOF_
  # ----
  # This defines which namespace the replication system uses
  # ----
  cluster name = $CLUSTER;

  # ----
  # Admin conninfo's are used by the slonik program to connect
  # to the node databases. So these are the PQconnectdb arguments
  # that connect from the administrators workstation (where
  # slonik is executed).
  # ----
  node 1 admin conninfo='dbname=$DBNAME1 host=$HOST1 user=$SLONY_USER';
  node 2 admin conninfo='dbname=$DBNAME2 host=$HOST1 user=$SLONY_USER';
```

```
# ----
# Initialize the first node. The id must be 1.
# This creates the schema "_test1" containing all replication
# system specific database objects.
# ----
init cluster ( id = 1, comment = 'Node 1' );

# ----
# The pgbench table history does not have a primary key or
# any other unique constraint that could be used to identify
# a row. The following command adds a bigint column named
# "_Slony-I_test1_rowID" to the table. It will have a default
# value of nextval('"_test1".sl_rowid_seq'), be unique and not
# null. All existing rows will be initialized with a number.
# ----
table add key ( node id = 1, full qualified name = 'public.sds10' );

# ----
# The Slony replication system organizes tables in sets. The
# smallest unit another node can subscribe is a set. Usually the
# tables contained in one set would be all tables that have
# relationships to each other. The following commands create
# one set containing all 4 pgbench tables. The "master" or origin
# of the set is node 1.
# ----


create set ( id = 1, origin = 1, comment = 'All tables on Master 1' );
```

```
  set add table ( set id = 1, origin = 1,
                                id = 1, full qualified name = 'public.sds10',
                                comment = 'Scientific Data Table' );



  # ----
  # Create the second node, tell the two nodes how to connect to
  # each other and that they should listen for events on each
  # other. Note that these conninfo arguments are used by the
  # slon daemon on node 1 to connect to the database of node 2
  # and vice versa. So if the replication system is supposed to
  # use a separate backbone network between the database servers,
  # this is the place to tell it.
  store node ( id = 2, comment = 'Node 2' );
  store path ( server = 1, client = 2,     conninfo='dbname=$DBNAME1              host=$HOST1
user=$SLONY_USER');
  store path ( server = 2, client = 1,
                                conninfo = 'dbname=$DBNAME2 host=$HOST2
user=$SLONY_USER');
  store listen ( origin = 1, provider = 1, receiver = 2 );
  store listen ( origin = 2, provider = 2, receiver = 1 );
_EOF_
```

Anexo 2. Slonyk 2.0

```
#!/bin/sh

CLUSTER=usercluster
DBNAME1=test00
DBNAME2=test00copy
HOST1=127.0.0.1

SLONY_USER=postgres
PGBENCH_USER=root

slonik <<_EOF_
  # ----
  # This defines which namespace the replication system uses
  # ----
  cluster name = $CLUSTER;
  # ----
  # Admin conninfo's are used by the slonik program to connect
  # to the node databases. So these are the PQconnectdb arguments
  # that connect from the administrators workstation (where
  # slonik is executed).
  # ----
  node 1 admin conninfo='dbname=$DBNAME1 host=$HOST1 user=$SLONY_USER';
  node 2 admin conninfo = 'dbname=$DBNAME2 host=$HOST2 user=$SLONY_USER';
```

```
  # ----
  # Node 2 subscribes set 1
  # ----
  subscribe set ( id = 1, provider = 1, receiver = 2, forward = no );
_EOF_
```

Anexo 3 Herramienta de Importación HDF2PDB

```
#include <hdf5.h>
#include "libpq-fe.h"
int main(int argc, char* argv[]) {
 hid_t file_id, dataset_id; /* identifiers */
 typedef struct {
 char *description;
 double dset_data[5][175][365];
 } TestIntData;
 TestIntData data;
 char *HDF_file = "tovs5day.h5";
 char *dataset = "/Data-Set-10";
 char stmt [ 255 ];
 herr_t status;
 file_id = H5Fopen(HDF_file, H5F_ACC_RDWR, H5P_DEFAULT);
 dataset_id = H5Dopen(file_id, dataset);
 data.description = "Datos de prueba";
    status = H5Dread(dataset_id, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL,
H5P_DEFAULT, data.dset_data);
 PGconn *psql;
 PGresult *res;
 psql = PQconnectdb("hostaddr='10.12.16.3' port='5432' dbname='mydb' user='postgres' ");
```

```
if (!psql)
fprintf(stderr, "Error de conexión");

int i, j, k;

for (i=0; i<4;i++)
for (j=0; j<174;j++) // latitud
for(k=0; k<365;k++) // longitud
{
sprintf(stmt, "SELECT insertdata(%d,%d,%f,%d)",j,k,data.dset_data[i][j][k],i );
printf(stmt);
res = PQexec(psql,stmt);
}s

/* Close the dataset */
status = H5Dclose(dataset_id);

/* Close the file */
status = H5Fclose(file_id);
}
```