UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES

A C A T L Á N

PROCESAMIENTO PARALELO EN BASES DE DATOS

TESIS

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN MATEMÁTICAS APLICADAS Y COMPUTACIÓN

PRESENTA:

JUAN PABLO CORREA CONTRERAS

ASESOR: M. EN C. JUDITH JARAMILLO LÓPEZ

MARZO DE 2005

Introducción

Las bases de datos surgen a partir de la necesidad de guardar información relevante de las diferentes organizaciones en un lugar seguro, confiable y de fácil acceso para realizar consultas. Con el aumento de las computadoras en el mercado, cada vez son más accesibles para las empresas y así se logra implementar en ellas las bases de datos relacionales como una herramienta de trabajo.

El software manejador de bases de datos fue tomando cada vez más importancia dentro de las empresas y controlando la mayoría de la información manejada en éstas; su gran éxito se ha dado de tal manera que se ha tenido que perfeccionar su funcionamiento, para lo cual hubo que definir nuevos métodos para buscar, borrar y actualizar los datos almacenados de una manera más rápida y eficiente, este es uno de los puntos de los cuales se van a tratar en este trabajo.

La manera en que se puede mejorar la respuesta de una base de datos es haciendo uso óptimo del hardware actual, con esto podemos comparar lo que es una base de datos trabajando en diferentes arquitecturas haciendo uso de procesos en serie y/o paralelo.

Para poder utilizar el procesamiento en paralelo se necesitan ciertos requerimientos mínimos los cuales son más costosos que los normales, pero permiten manejar una cantidad mayor de información; estas arquitecturas tanto de hardware como de software serán analizadas en el primer capítulo.

No todas las versiones de los manejadores de bases de datos tienen la capacidad de realizar procesamiento paralelo, en algunas de ellas la herramienta que permite realizar estas acciones se tiene que instalar por separado, por lo cual en este trabajo se va a utilizar específicamente la herramienta de Oracle, la cual es Oracle Parallel Server. Esto es debido a que

es la herramienta a la que se tiene acceso en el centro de cómputo y es el manejador con el cual tengo mayor experiencia.

Después se analizarán las operaciones que se pueden paralelizar para poder obtener un rendimiento máximo del equipo con el que se esté trabajando así como de la base de datos, lo cual se discutirá en el segundo capítulo.

En el tercer capítulo se realizará una introducción a la herramienta utilizada para probar dichas arquitecturas, esta herramienta es "*Oracle Parallel Server*", del cual se explicará su arquitectura y funcionamiento basándose en los diferentes puntos ilustrados en los capítulos anteriores.

Por último, en el cuarto capítulo, se mostrará como es que se puede implementar y migrar a un ambiente como el propuesto para poder realizar algunas pruebas sobre el rendimiento obtenido de las aplicaciones bajo la arquitectura SMP.

Hipótesis

Al aplicar el procesamiento en paralelo en las bases de datos se optimizan tiempos de respuesta, por lo cual la información se obtiene en un menor tiempo aumentando la productividad y utilizando de manera óptima los equipos de cómputo.

Objetivo

Utilizar la tecnología de punta que se maneja en las herramientas de las bases de datos, de tal manera que el resultado sea la constante evolución de los sistemas computacionales que se basan en explotar información de dicha fuente.

Arquitectura necesaria para el procesamiento en paralelo.

En este capítulo se hablará sobre los diferentes requerimientos que se necesitan en las diversas estructuras en las que se puede implementar el procesamiento en paralelo en bases de datos. Estos requerimientos son tanto de hardware como de software; las arquitecturas de las que se hablará son:

- SMP
- Clustered
- MPP
- NUMA

También se explicarán las ventajas y desventajas de la utilización de cada una de estas arquitecturas.

Acerca del procesamiento en paralelo.

El procesamiento paralelo es la forma en que se puede dividir una tarea en varias subtareas o procesos, los cuales son ejecutados simultáneamente por un equipo de cómputo y así intentar que el resultado se obtenga en un tiempo menor, para ello se requiere de al menos dos procesadores. Al aplicar esta técnica en el software manejador de bases de datos se tienen que tomar en cuenta muchas posibilidades como si se tratara de llevar el procesamiento paralelo a una máquina que cuenta con un solo procesador; esto no procede ya que de cualquier forma el proceso se realizaría de forma lineal aunque las *tareas* se pudieran ejecutar por separado, lo que resulta como una pérdida de tiempo al contar con los procesos en la cola esperando a ser ejecutados, como se ve en la figura 1.1.

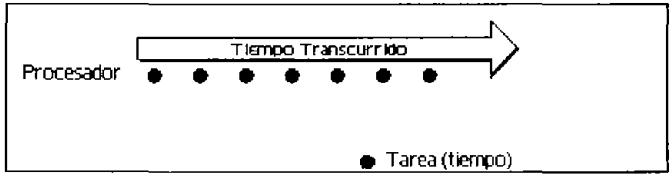


Figura 1.1. Ejecución de procesos independientes con un solo procesador.

Otro punto que se tiene que considerar es la manera en que se paralelizan los procesos, ya que estos se pudieran llegar a ejecutar por separado, pero si cada uno de estos procesos necesita datos del proceso anterior se pierde la ventaja de ejecución en paralelo; porque para que se ejecute un proceso se tiene que esperar a que se hayan ejecutado los anteriores, como se muestra en la Figura 1.2. A este tipo de procesos se les conoce como procesos dependientes o incluyentes. *[BAM99]*

A diferencia de si se cuenta con la arquitectura necesaria para procesar en paralelo cada procesador se ocupa de realizar un proceso diferente, esto se refleja en el tiempo de respuesta

que se tiene utilizando este tipo de arquitectura. En la figura 1.3, se puede observar la diferencia del tiempo de respuesta con respecto al proceso lineal y con procesos dependientes.

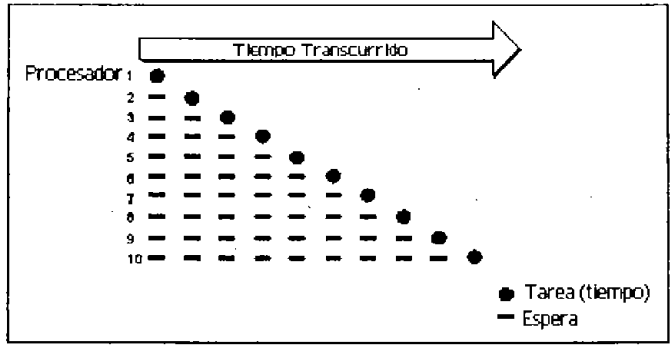


Figura 1.2. Ejecución de procesos dependientes en una arquitectura con varios procesadores.

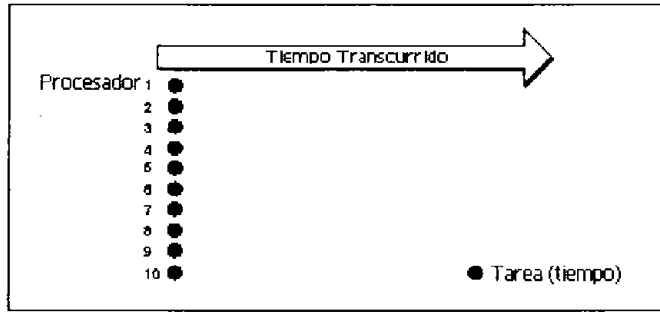


Figura 1.3. Procesamiento en paralelo. Cada procesador ejecuta un proceso independiente.

Características de un sistema de cómputo paralelo.

Este tipo de sistema cuenta con las siguientes características:

- Cada procesador puede realizar *tareas concurrentes.*
- Algunas *tareas* deben ser *sincronizadas.*
- Los nodos normalmente comparten recursos como discos, datos y otros dispositivos.

Las arquitecturas para procesamiento paralelo deben soportar alguno del siguiente hardware:

- Procesamiento en grupo y masivo en paralelo, en donde cada procesador tiene su propia memoria.
- Equipos con una sola memoria, también conocidas como multiprocesamiento simétrico, en los cuales los procesadores utilizan los mismos recursos de memoria. *[BAU99]*

Metas de un procesamiento paralelo.

Las metas del procesamiento paralelo se pueden medir con base a dos parámetros importantes:

- Reducir tiempos (velocidad).
- Permitir o aprovechar (escalabilidad).

Velocidad es el tiempo de respuesta, es decir, que tanto hardware necesito para ejecutar el mismo proceso en un tiempo menor que el sistema original. Esta aceleración de procesos está dada directamente por el hardware que se utilice. Como se muestra en la figura 1.4.

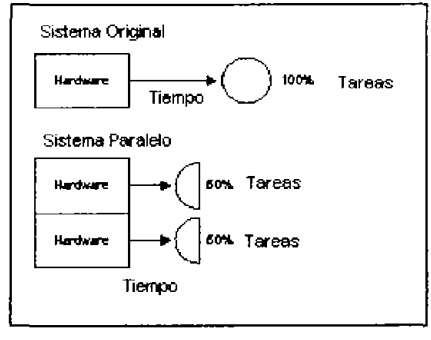


Figura 1.4. Relación de aumento de hardware para acelerar el tiempo de respuesta al realizar el mismo número de procesos.

La escalabilidad está relacionada con el factor de qué tanto trabajo más se desea realizar en el mismo periodo de tiempo. Como se muestra en la figura 1.5.

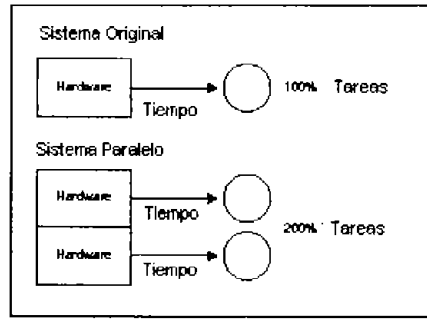
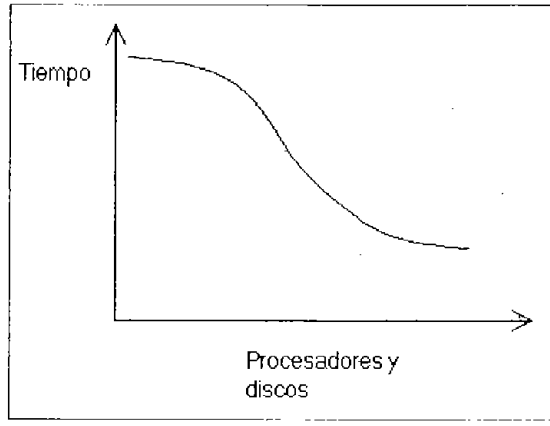


Figura 1.5. Aumento de *tareas* que se realizan en el mismo tiempo.

La ganancia de velocidad y la escalabilidad son dos aspectos importantes en el estudio del paralelismo. La ganancia de velocidad se refiere a la ejecución en menos tiempo de una *tarea* dada mediante el incremento del grado de paralelismo. La escalabilidad se refiere al manejo de transacciones más largas mediante el incremento del grado de paralelismo. *[BAU99]*

Considerando un sistema paralelo con un cierto número de procesadores y discos que está ejecutando una aplicación de *base de datos*. Supóngase ahora que se incrementa el tamaño del equipo donde se encuentra el sistema añadiéndole más procesadores, discos y otros componentes. El objetivo es realizar el procesamiento de la *tarea* en un tiempo inversamente proporcional al número de procesadores y discos del sistema, como se muestra en la gráfica 1.1.



Gráfica 1.1 El tiempo es inversamente proporcional al número de procesadores y discos de un sistema

La escalabilidad está relacionada con la capacidad de procesar *tareas* más largas en el mismo tiempo mediante el incremento de los recursos del sistema. La manera de medir el tamaño de las *tareas* da lugar a dos tipos de escalabilidad relevantes en los sistemas paralelos de bases de datos:

- En la **escalabilidad por lotes** aumenta el tamaño de la *base de datos*, y las *tareas* son trabajos más largos cuyos tiempos de ejecución dependen del tamaño de la *base de datos*. Recorrer una relación cuyo tamaño es proporcional al tamaño de la *base de datos* sería un ejemplo de tales *tareas*. Así, la medida del tamaño del problema es el tamaño de la *base de datos*. La escalabilidad por lotes también se utiliza en aplicaciones científicas, tales como la ejecución de una consulta con una resolución *n* veces mayor o la realización de una simulación N veces más larga.
- En la **escalabilidad de transacciones** aumenta la velocidad con la que se envían las transacciones a la *base de datos* y el tamaño de la *base de datos* crece proporcionalmente a la tasa de transacciones. Este tipo de escalabilidad es el relevante en los sistemas de procesamiento de transacciones en los que las transacciones son modificaciones pequeñas, por ejemplo, un abono o retiro de fondos

de una cuenta; y cuantas más cuentas se creen, más crece la tasa de transacciones. Este procesamiento de transacciones se adapta especialmente bien a la ejecución en paralelo, ya que las transacciones pueden ejecutarse concurrente e independientemente en procesadores distintos, y cada *transacción* dura más o menos el mismo tiempo, aunque crezca la *base de datos*.

La escalabilidad es normalmente el factor más importante para medir la eficiencia de un sistema paralelo de bases de datos. El objetivo del paralelismo en los sistemas de bases de datos pretende asegurar que la ejecución del sistema continuará realizándose a una velocidad aceptable, incluso en el caso de que aumente el tamaño de la *base de datos* o mediante el incremento del paralelismo proporciona a una empresa un modo de crecimiento más suave que el de reemplazar un sistema centralizado por una máquina más rápida (suponiendo incluso que esta máquina existiera).

Arquitecturas de Hardware.

El procesamiento en paralelo se enfoca en utilizar varios procesadores o computadoras para reducir el tiempo de respuesta que se necesita para concluir ciertas *tareas* o transacciones. Para dicho fin, se requiere de hardware especial el cual permita realizar este tipo de *tareas*. Se han desarrollado varias tecnologías que pueden trabajar con múltiples procesadores y/o CPU's. Las más comerciales son las siguientes:

- Multiprocesamiento simétrico (SMP).
- Agrupaciones (Cluster).
- Procesamiento paralelo masivo (*MPP*)
- Acceso a memoria no uniforme (NUMA)

Multiprocesamiento simétrico (SMP)

En una arquitectura de multiprocesamiento simétrico o SMP se cuenta con dos o más procesadores en un equipo, los cuales, comparten los mismos recursos (memoria y dispositivos de Entrada/Salida) esto es acompañado de un *bus* de alta velocidad para mejorar la comunicación entre estos recursos. También se requiere de un sistema operativo el cual esté diseñado para soportar múltiples procesadores y tener un administrador el cual se encargue de utilizar cada uno de los procesadores eventualmente. A esta arquitectura también se le conoce como arquitectura altamente unida a la memoria o de memoria compartida, este tipo de arquitectura lo podemos observar en la figura 1.6

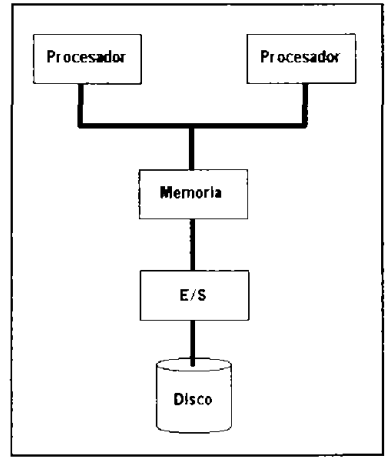


Figura 1.6. Arquitectura SMP típica con dos procesadores.

Cada procesador en un sistema SMP puede ejecutar un programa independientemente y cada procesador accesar a una parte separada de la memoria. Un sistema con un solo procesador puede correr en modo tiempo-compartido con el cual en realidad sólo puede realizar una *tarea* a la vez. Este completa la ilusión de trabajar con varias *tareas* al estar ciclando entre las diferentes *tareas* que se están ejecutando rápidamente. Por otro lado, un sistema SMP realmente trabaja con

varias *tareas* al mismo tiempo al asignar cada una a un procesador diferente, los cuales están dedicados a trabajar con cada *tarea*.

Como el trabajo de repartir las *tareas* entre los diferentes procesadores es responsabilidad del sistema operativo, este cambio es transparente para las aplicaciones, por lo que los beneficios de esta arquitectura se obtienen sin tener que realizar algún cambio en el *ORDBMS* como en la aplicación. Por otro lado se puede hacer que en un sistema SMP una *tarea* se asigne a varios procesadores y que estos trabajen cooperativamente para que su tiempo de ejecución sea más rápido que realizándolo con un solo procesador; aunque para poder realizar este tipo de operación se requiere que el software del *RDBMS* lo pueda soportar. *[GRE01]*

Ventajas

La arquitectura SMP es de las que más historia tiene, también es la que más ventas tiene en el mercado y, por lo tanto, más proveedores comerciales. Las arquitecturas SMP traen varias ventajas con ellas como lo son:

- Incrementar el rendimiento de un sistema; en realidad lo que se hace es aumentar el número de procesadores, también se puede agregar más memoria o discos según lo requiera el sistema.
- El software diseñado para ejecutarse en una arquitectura con un procesador trabaja de igual manera sin tener que realizar modificación alguna.
- Los sistemas SMP representan un buen camino visto del lado económico para obtener escalabilidad en cuyo caso con un sistema con un solo procesador no pueda alcanzar el rendimiento necesario. SMP es ideal siempre y cuando los requerimientos de escalabilidad se encuentren dentro de los límites tecnológicos del hardware.
- Esta arquitectura es madura y ampliamente utilizada en el mercado actual.

- Al utilizar una sola copia del sistema operativo corriendo en el sistema, la carga administrativa para controlar un sistema SMP es similar a la de controlar la de un solo procesador.

Desventajas

Con todo y sus buenos puntos, la arquitectura SMP trae con ella un par de desventajas significantes:

- La arquitectura SMP no tiene una buena escalabilidad cuando se trata de un número de procesadores alto.
- La arquitectura SMP no presenta una disponibilidad alta.

El número de procesadores que se pueden añadir a un sistema SMP es limitado, porque todos los procesadores comparten la misma memoria. Si se añaden demasiados procesadores se tendría como consecuencia un *cuello de botella* en el acceso a memoria como en la utilización del *bus* de comunicaciones. El número máximo de procesadores que pueden ser configurados en un sistema SMP varia dependiendo de cada proveedor y/o sistema operativo utilizado. Por ejemplo, Windows NT soporta de dos a un máximo de sólo cuatro procesadores, mientras los sistemas SMP con sistema operativo Unix soportan desde dos a un máximo de 64 procesadores.

Otra de las consideraciones que se deben de tomar en cuenta para un sistema SMP es que, como solamente se está ejecutando una copia del sistema operativo, este representa un punto de falla. Por otro lado, si se presenta algún problema con un dispositivo (hardware) esto haría que el sistema estuviera fuera de operación; por ello es que los sistemas SMP no son una buena elección cuando lo que se requiere es una gran disponibilidad y esta es vital. *[MAH00]*

Clustered Systems

En la arquitectura de clustered, un pequeño grupo de nodos están interconectados y comparten dispositivos de almacenamiento. Cada nodo tiene un sistema independiente con su propio procesador y memoria, también corre su propia copia de sistema operativo. Cada nodo puede tener un solo procesador o ser un equipo SMP. En la figura 1.7 se ilustra esta arquitectura con dos nodos que tienen un equipo SMP con dos procesadores cada uno. La arquitectura **clustered** es también conocida como una arquitectura poco acoplada, porque se permite una configuración distinta tanto como para el procesador como para la memoria en cada uno de los nodos.

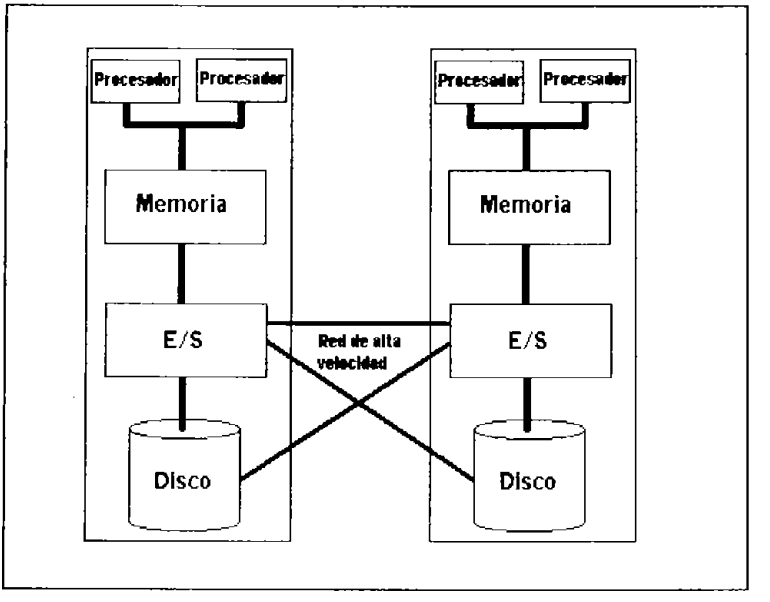


Figura 1.7. Arquitectura Clustered con dos equipos SMP

Ventajas

Los sistemas con arquitectura Clustered tienen ventajas sobre otras como:

- Con Clusters se tiene un mejor rendimiento y escalabilidad que con cualquier sistema con arquitectura SMP. La escalabilidad de un sistema Clustered puede ser incrementada de dos maneras: incrementando el número de procesadores en cada nodo y/o incrementando el número de nodos que se tienen en los clusters.
- Esta arquitectura provee de una opción no muy cara para obtener escalabilidad ya que utiliza las arquitecturas más utilizadas y tecnología de redes estándar.
- También ofrece una gran disponibilidad. A menudo las redes entre los nodos está hecha con conexiones duales para obtener redundancia. Si un nodo del cluster falla, otros nodos dentro del mismo cluster siguen operando, y los usuarios conectados al nodo con problemas pueden ser redistribuidos entre los nodos que siguen funcionando.

Desventajas

Las desventajas que presentan los sistemas con arquitecturas de cluster son:

- La administración y monitoreo de un sistema de cluster necesita de software adicional. Por ejemplo, para administrar clusters IBM RS/6000 utilizando un sistema operativo AIX, también se necesita de un software llamado HACMP (High Availability Clusters Multiprocessing) del mismo IBM.
- La administración de un sistema de clusters es una *tarea* más compleja que manejar un sistema con un solo nodo SMP. La complejidad aumenta dependiendo del número de nodos utilizados.

- Comparado con un sistema SMP, un sistema de clusters requiere de un mayor esfuerzo para coordinar las *tareas* en los diferentes nodos, así como para balancear la carga de trabajo a través de los diferentes nodos. *[MAH00]*

Procesamiento Masivamente Paralelo, Massively Parallel Processing (MPP)

En la arquitectura de procesamiento masivamente paralelo (*MPP*, del inglés Massively Parallel Processing) varios nodos están conectados por medio de una red de alta velocidad. Cada nodo tiene su propio procesador y memoria, en la figura 1.8 se muestra esta arquitectura utilizando cuatro nodos. Por que cada nodo tiene su propia memoria, los sistemas *MPP* son conocidos como sistemas de memoria distribuida. Cada nodo corre su propia copia de sistema operativo, y la comunicación entre los nodos es mediante la transmisión de mensajes. El número de nodos que puede ser configurado en una arquitectura *MPP* varia desde dos hasta cientos. Así como en los sistemas de cluster, un nodo puede tener un solo procesador o pueden ser equipos SMP. Los sistemas *MPP* fueron hechos originalmente para realizar una alta paralelización de operaciones científicas, pero ahora están siendo utilizadas para aplicaciones de data warehuosing.

Los sistemas MPP y de cluster tienen muchas similitudes desde un punto de vista de su arquitectura. Ambos utilizan arquitectura de memoria distribuida, y también en ambos cada nodo corre su propia copia de sistema operativo. Estas arquitecturas utilizan transmisión de mensajes para realizar la comunicación entre los distintos nodos y requieren de una programación especial para explotar apropiadamente el paralelismo. La diferencia fundamental entre los sistemas *MPP* y de cluster es que el número de nodos que se pueden configurar con un sistema *MPP* es mucho mayor comparado con los que se pueden tener en un sistema de cluster. Otra diferencia es la interconexión que existe en estas arquitecturas. *[MAH00]*

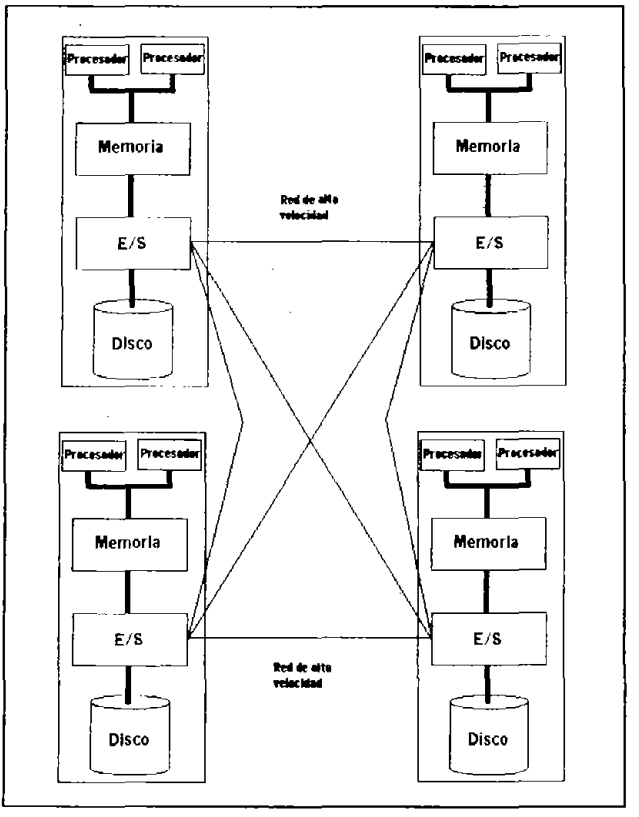


Figura 1.8. Arquitectura *MPP* con cuatro nodos SMP.

Ventajas

La ventaja principal de un sistema *MPP* es que con ésta no se presenta el *cuello de botella* en el acceso a memoria que es lo que limita la escalabilidad de un sistema SMP, porque cada nodo en un sistema *MPP* tiene su propia memoria. Mientras los nodos en un sistema *MPP* pueden ser equipos SMP, ellos típicamente sólo tienen unos pocos procesadores, manteniéndolo por debajo de los límites de escalabilidad del equipo SMP.

Los sistemas *MPP* son localizados en grandes aplicaciones de data warehousing. Un sistema *MPP* grande puede soportar varios miles de usuarios y unas 10,000 o más transacciones por minuto.

Desventajas

Los sistemas *MPP* tienen dos grandes desventajas:

- Dado que la memoria en cada nodo de un *MPP* está separada, se necesitan dar de alta características especiales en el sistema operativo para poder implementar la comunicación entre los distintos nodos. En sistemas *MPP*, cuando un nodo necesita información contenida en un nodo distinto, se realiza un intercambio de mensajes entre estos nodos para poder obtener dicha información.
- Los sistemas *MPP* son más difíciles de administrar que los SMP, porque cada nodo en un sistema *MPP* corre su propia copia de sistema operativo y también su propia copia del software de la *base de datos*.

Acceso de memoria no uniforme, Non uniform memory access (NUMA)

Las computadoras con arquitectura de acceso de memoria no uniforme (NUMA, del ingés Non Uniform Memory Access) comenzaron a mediados de la década de los 90's. Brindan una mayor velocidad que los SMP al ligar múltiples componentes SMP por medio de memoria distribuida, como se muestra en la figura 1.9. Como en los sistemas *MPP*, éstos proveen escalabilidad de memoria y subsistemas de E/S con la adición de utilizar múltiples computadoras. De cualquier modo, una sola copia del sistema operativo controla a todos los equipos y hace uso de un esquema de direccionamiento de coherencia caché para conservar la sincronía de los datos. El acceso a la memoria entre los diferentes nodos se mide en cientos de microsegundos, lo cual es

más rápido que el ir a leer un disco en una arquitectura con configuración *MPP* o cluster, aunque es más lento que leer en memoria local como en un sistema SMP. *[GRE01]*

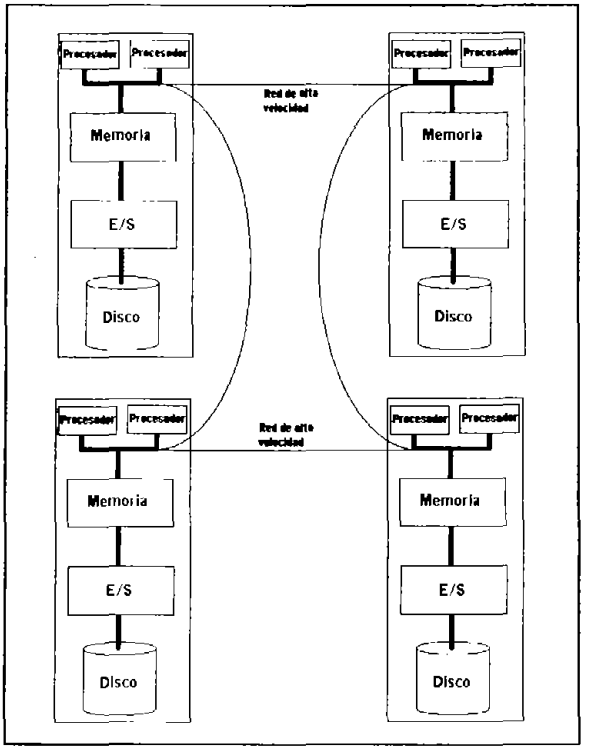


Figura 1.9. Arquitectura Non Uniform Memory Access (NUMA).

Ventajas

La arquitectura NUMA fue desarrollada para que los sistemas SMP pudieran soportar una mayor escalabilidad y por lo tanto cuentan con las siguientes ventajas:

- Provee una mayor escalabilidad a la arquitectura SMP.
- Las aplicaciones no tienen que ser acondicionadas para esta nueva arquitectura.

Desventajas

Las desventajas a las que se está sujeto con este tipo de arquitectura son:

- Mientras el acceso a memoria local es rápido, el acceso a memoria remota no lo es tanto. Un alto empleo de memoria no local puede hacer que el sistema funcione más lento.
- El rendimiento no depende solamente de la red de alta velocidad, sino de una buena afinación del sistema operativo. Para poder explotar esta arquitectura el software de *base de datos* utilizado debe ser enfocado a la misma.
- Igual que con los sistemas SMP, un punto de falla puede ser el sistema operativo.

Arquitectura de Software de la Base de Datos

Al implementar el paralelismo al software de *base de datos* se establecieron tres arquitecturas diferentes, comúnmente conocidas como:

- Todo compartido
- Disco compartido
- Nada compartido

Lo compartido se refiere al hecho de compartir los recursos de disco y memoria por múltiples equipos.

Para poder implementar una de estas tres arquitecturas de software se requiere de una buena base formada por una arquitectura de hardware que soporte el procesamiento en paralelo. Algunas arquitecturas de software son mejores para cierta arquitectura de hardware. Por ejemplo, una arquitectura de software de todo compartido trabaja de forma natural con una arquitectura de hardware de tipo SMP, porque todos los procesadores de un sistema SMP comparten la misma

memoria y disco. Otras combinaciones no son muy buenas, en el caso de implementar una arquitectura de todo compartido en una plataforma *MPP*, no es una buena opción, porque la arquitectura *MPP* está basada en una arquitectura de memoria distribuida. La implementación de la abstracción de software sobre memoria compartida con nodos individuales en un sistema *MPP* sería difícil y el rendimiento sería muy bajo. *[BAM99]*

Todo compartido

En una arquitectura de *base de datos* de tipo todo compartido, todos los procesadores comparten la misma memoria y discos. En la figura 1.10 se ilustra dicho sistema. Una copia de el sistema operativo y una copia del software de la *base de datos* corre en este sistema. La memoria compartida permite la coordinación eficiente entre los procesos del *RDBMS*. En esta arquitectura, es relativamente sencillo implementar el paralelismo en sentencias SQL, porque el sistema operativo aloja automáticamente las consultas y subconsultas a los procesadores disponibles. La arquitectura de software de todo compartido es ampliamente utilizada en hardware de tipo SMP. La arquitectura de hardware de tipo NUMA también es compatible con la arquitectura de software de todo compartido.

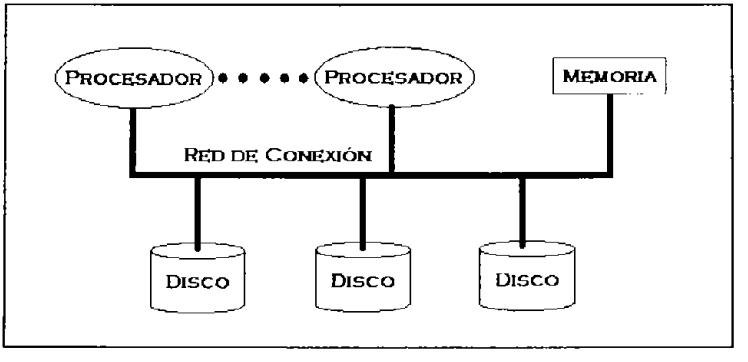


Figura 1.10. Arquitectura con todo compartido.

Todos los proveedores de *RDBMS* dan soporte para la arquitectura de todo compartido. Oracle soporta la ejecución en paralelo en la arquitectura de software de todo compartido. Esta

arquitectura tiene limitaciones de escalabilidad y disponibilidad fundamentalmente por la arquitectura de hardware. Por ejemplo, con el hardware de SMP, los cuellos de botella de memoria y disco limita el número de procesadores que puede ser utilizados en la arquitectura de todo compartido. Adicionalmente, se cuentan con dos puntos de falla porque solo una copia del sistema operativo y una copia del software de la *base de datos* son las que se están ejecutando.

Disco compartido

En la arquitectura de disco compartido, cada nodo tiene su propio CPU y memoria. Los dispositivos de almacenamiento (discos) son conectados utilizando una red de alta velocidad y son compartidos por todos los nodos. Cada nodo tiene acceso a cualquier disco. Cada nodo también corre su propia copia de sistema operativo y software de *base de datos.* Lógica y físicamente sólo hay una **base de datos**, distribuida en todos los discos, los cuales son accesados por todos los nodos. En la figura 1.11 se muestra la arquitectura de discos compartidos.

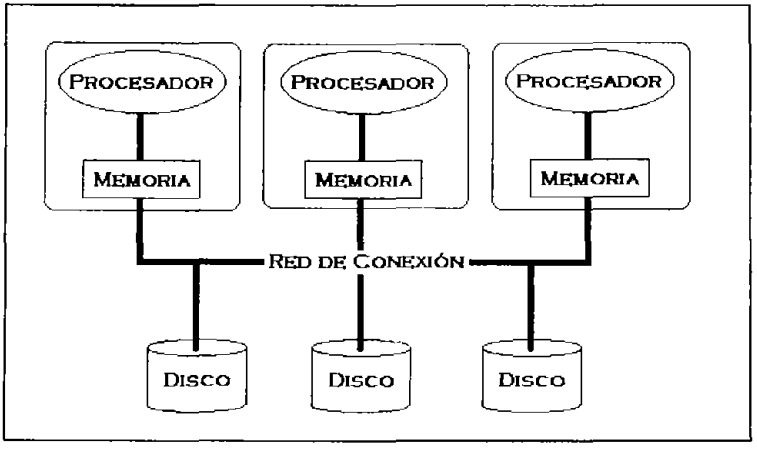


Figura 1.11. Arquitectura de discos compartidos.

El objetivo de compartir los discos es el de eliminar los cuellos de botella en la lectura de la información en los sistemas. Para incrementar el rendimiento o para soportar un mayor número de usuarios, se pueden añadir nodos y discos. Dado a que se están añadiendo nodos, no se cuenta con cuellos de botella en memoria a los cuales los sistemas SMP están expuestos. Los sistemas de disco compartido y nada compartido tienen esta ventaja de escalabilidad sobre los sistemas con la arquitectura de todo compartido y, consecuentemente, frecuentemente son utilizadas en bases de datos de gran tamaño.

Nada compartido

En la arquitectura de nada compartido, cada nodo es independiente y tiene su propia memoria y disco. Cada nodo corre su propia copia de sistema operativo y su copia del software de *base de datos.* Una red de alta velocidad es utilizada para conectar a los diferentes nodos. La figura 1.12, muestra como los equipos bajo una arquitectura de nada compartido no comparten memoria ni discos entre los nodos. La arquitectura de nada compartido es posible emplearla en sistemas de **cluster** y ***MPP***.

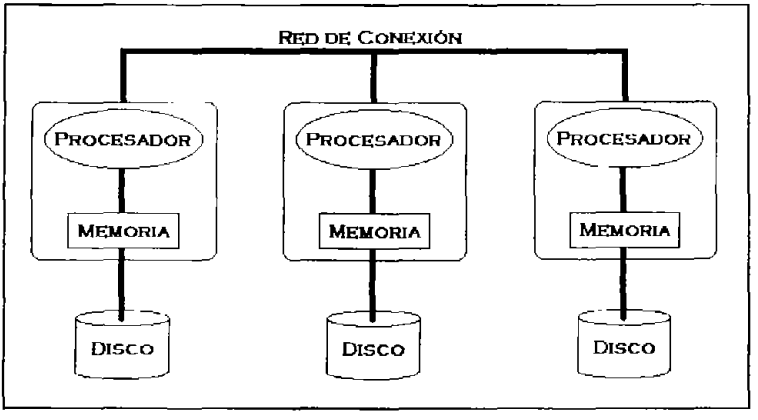


Figura 1.12. Arquitectura de nada compartido con tres nodos.

Las bases de datos en una arquitectura de nada compartido están particionadas, o divididas, entre los nodos. Cada nodo tiene acceso directo a solo una parte de la *base de datos*, referenciada como partición local, que está almacenada en el disco local. Los nodos no tienen acceso directo a los datos en particiones no locales.

En un ambiente de nada compartido, las consultas de la *base de datos* que accesan sólo a los datos que se encuentran en la partición local se ejecutan mucho más rápido que los que requieren datos de una partición no local. Cuando la información se requiere de una partición no local, el *RDBMS* transmite la consulta al nodo remoto dueño de la partición en cuestión. Ese nodo realiza la ejecución de la consulta, obtiene los datos, y envía el resultado al nodo correspondiente.

Ejecución en paralelo II

Una *tarea* se puede llevar a cabo de distintas formas, una de ellas es en paralelo. Uno de los puntos de los cuales se va a hablar es como funciona y ejecuta una *tarea* en paralelo; también cómo es que se realizan las consultas (*queries*) en paralelo y cuáles son las operaciones que se pueden paralelizar.

Un tema muy importante al hablar de paralelismo es el grado de paralelismo con el cual se van a ejecutar las sentencias, esto se relaciona directamente con el número de procesos que se ejecutarán para realizar dicha *tarea*.

Introducción a la ejecución en paralelo.

A partir del presente capítulo se explicarán algunas de las características del procesamiento en paralelo con el manejador de bases de datos Oracle versión 8.1.7, el cual es capaz de realizar este tipo de procesamiento y a su vez es una de las herramientas más utilizadas en el mercado tanto nacional como internacional. Actualmente en el mercado existen otros manejadores que también tienen la capacidad de realizar el procesamiento paralelo como: Informix, Sybase, Progress, DB2, etc; pero para fines de este trabajo solamente se utilizó Oracle, por ser el que tiene el Centro de Cómputo de nuestra universidad y al cual tengo acceso para realizar las pruebas.

El procesamiento en paralelo divide una *tarea* en varios subprocesos que son ejecutados en diferentes procesadores y el resultado obtenido es que la *tarea* se realiza en un tiempo menor. Esto sólo se puede realizar en *tareas* que pueden ser divididas ya que hay algunas *tareas* que por naturaleza no se pueden dividir, pero cuando se logra hacer esta división el procesamiento paralelo resulta un método muy eficaz.

Los sistemas paralelos mejoran la velocidad de procesamiento y de entrada/salida (E/S) mediante la utilización del CPU y discos en paralelo. Cada vez son más comunes las máquinas paralelas, lo que hace que cada vez sea más importante el estudio de los sistemas paralelos de bases de datos relacionales. La fuerza que ha impulsado a los sistemas paralelos de bases de datos relacionales ha sido la demanda de aplicaciones que han de manejar bases de datos extremadamente grandes (del orden de terabytes, esto es, 10^{12} bytes) o que tienen que procesar un número enorme de transacciones por segundo (del orden de miles de transacciones por segundo). Los sistemas de bases de datos centralizados o cliente-servidor no son suficientemente potentes para soportar tales aplicaciones. *[MAH00]*

¿Cómo funciona un proceso paralelo?

En el procesamiento paralelo se realizan muchas operaciones simultáneamente, mientras que en el procesamiento lineal los distintos pasos computacionales han de ejecutarse secuencialmente. Una máquina con un nivel bajo de paralelismo consiste en un pequeño número de potentes procesadores; una máquina masivamente paralela utiliza varios procesadores más pequeños. Hoy en día, la mayoría de las máquinas de comerciales ofrecen un cierto grado de paralelismo: son comunes las máquinas con dos o cuatro procesadores. Las computadoras masivamente paralelas se distinguen de las máquinas paralelas simétricas porque son capaces de soportar un grado de paralelismo mucho mayor. Ya se encuentran en el mercado computadoras paralelas con cientos de CPU y discos.

En las figuras 2.1 y 2.2 se puede observar el contraste del tiempo requerido para realizar una *tarea* en forma lineal y en forma paralela.

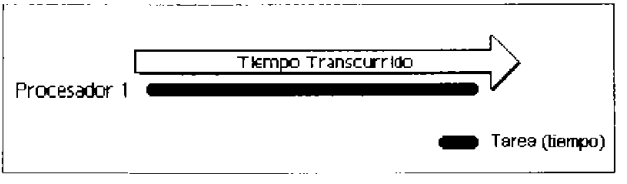


Figura 2.1. Procesamiento lineal en el que se ejecuta un solo proceso.

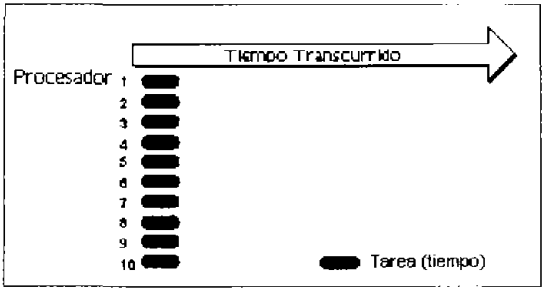


Figura 2.2. Procesamiento en paralelo en el que el proceso es divido en *tareas* más pequeñas que se ejecutan en procesadores diferentes.

Para medir el rendimiento de los sistemas de bases de datos existen dos medidas principales:

- La **productividad**. Número de *tareas* que pueden completarse en un intervalo de tiempo determinado.
- El **tiempo de respuesta**. Cantidad de tiempo que se necesita para completar una única *tarea* a partir del momento en que se envíe.

Un sistema que procese un gran número de pequeñas transacciones puede mejorar la productividad realizando muchas transacciones en paralelo. Un sistema que procese transacciones largas puede mejorar el tiempo de respuesta, así como la productividad, realizando en paralelo las distintas subtareas de cada *transacción. [BAM99]*

Consultas en paralelo.

El primer proceso que se paralelizó en Oracle fue la consulta de datos, y se manejaba como una herramienta separada del manejador de *base de datos*. Con Oracle este producto salió con la versión 7.1 y se llamaba *Oracle Parallel Query Option* (PQO), posteriormente fue integrado al paquete estándar del producto.

Las consultas en paralelo pueden reducir significativamente el tiempo de respuesta de consultas robustas, pero esto no aplica a todas las consultas. Para poder paralelizar una consulta se deben de tomar en cuenta los siguientes puntos:

- Por lo menos una de las *tablas* se va a consultar por completo, lo que se conoce como un "*Full Scan*", o el acceso a un *índice* está en un rango que se encuentre particionado.

- Si la consulta es sobre una *tabla* a la cual se le hace un "*Full Scan*" se tiene que especificar que se tiene que paralelizar en esta *tabla* mediante un *hint*[1] o la definición de la *tabla* debe de tener especificado el parámetro de paralelización.
- Si la consulta es por medio del un *índice* particionado, se debe de declarar que se paralelice en dicho *índice* en el *hint* de la consulta o el *índice* debe ser creado especificando el parámetro de paralelización.

Oracle puede paralelizar consultas simples, subconsultas, instrucciones definidas dentro del Lenguaje de Manipulación de Datos (DML, del inglés Data Manipulation Language) e instrucciones definidas dentro del Lenguaje de Definición de Datos (DDL, del inglés Data Definition Language). También hay que tomar en cuenta las siguientes restricciones al trabajar con las consultas en paralelo:

- Si una instrucción *DML* o *DDL* hace referencia a un objeto remoto, esa parte de la instrucción no puede ser paralelizable.
- Las consultas en *tablas* anidadas no pueden ser paralelizables. *[LEV99]*

Operaciones paralelizables

En Oracle 8.1.7 las siguientes operaciones son paralelizables:

Consultas

- Recorrido completo de una *tabla* (*Full scan*).
- Hash *joins*.
- Ligas con ciclos anidados.
- Agrupaciones (group by). Cláusula de agrupamiento de datos.

[1] Hint. Para cada sentencia SQL se genera un plan de ejecución, el cual indica la serie de pasos que se tienen que hacer para poder ejecutar la sentencia. En algunas ocasiones, se puede sugerir el camino para optimizar dicha sentencia. A esta sugerencia se le llama

- SELECT DISTINCT . Selecciona los datos que sean distintos.
- Sort merge *join*.
- Condiciones con "not in". No se encuentra en o no pertenece a.
- Uniones ("union" y "union all"). Operadores de unión entre dos consultas.
- Aggregation.
- Ordenaciones (order by). Cláusula de ordenamiento de datos.

DDL

- CREATE TABLE ... [AS SELECT]. Crea una *tabla*.
- ALTER INDEX ... REBUILD. Modifica un *índice*.
- CREATE INDEX. Crea un *índice*.
- ALTER TABLE ... MOVE PARTITION. Mueve una partición.
- ALTER TABLE ... SPLIT PARTITION. Divide o fragmenta una partición.
- ALTER TABLE ... COALESCE PARTITION. Desfragmenta una partición.
- ALTER INDEX ... REBUILD PARTITION. Reconstruye una partición.
- ALTER INDEX ... SPLIT PARTITION. (Solo si la partición del *índice* que se va a separar es utilizable).

DML

- *UPDATE*. Actualiza los datos de un renglón en una *tabla*.
- *DELETE*. Borra renglones de una *tabla*.
- *INSERT* ... SELECT. Inserta datos en una *tabla*.

OTRAS

- Parallel load. Opción utilizada por la herramienta Loader de Oracle para realizar cargas de datos en paralelo.
- Parallel replication. Opción utilizada para realizar duplicados en forma paralela.

- Parallel recovery. Opción utilizada para realizar recuperaciones en forma paralela. *[LEV99]*

Grado de paralelismo

Las dos cosas más importantes a afinar para utilizar una *base de datos* en paralelo son: la apropiada distribución de entradas y salidas y el grado de paralelismo. Especificar el grado de paralelismo es particularmente trivial, pero requiere de mucho trabajo de análisis para poder definirlo adecuadamente. Primero hay que analizar los siguientes factores:

- La capacidad de los procesadores del sistema. El número y capacidad de procesadores afecta directamente al número de procesos que un *query* debe de correr.
- La capacidad del sistema para manejar una gran cantidad de procesos. Algunos sistemas operativos pueden manejar muchos procesos simultáneos; otros son limitados.
- La carga del sistema. Si el sistema actualmente se encuentra trabajando al 100% de su capacidad, entonces el grado de paralelismo no va a tener mucho efecto. Si el sistema está al 90%, se puede llegar a sobre cargar el sistema.
- La cantidad de queries procesados en el sistema. Si muchas de las operaciones son actualizaciones, pero algunos queries son críticos se podrían añadir procesos para las consultas.
- La capacidad de entradas/salidas del sistema. Dependiendo de la configuración de los discos y de la carga se puede ver si se podría manejar un mayor número de consultas en paralelo.
- El tipo de operaciones. Las operaciones que más se benefician son las que están realizando muchos *full scans* u ordenamientos. *[WHA97]*

Configurando el grado de paralelismo

Una vez que Oracle (versión 8.1.7) va a ejecutar una instrucción SELECT en paralelo, el grado de paralelismo es determinado de la siguiente manera:

1) Oracle obtiene la especificación del grado y las *instancias* de la definición de todas las *tablas* e *índices* involucrados en el *query* y selecciona el valor máximo encontrado en estas especificaciones.
2) A continuación revisa la sentencia para ver si encuentra algún *hint* con la instrucción *PARALLEL* o *PARALLEL_INDEX*. Si dicho *hint* es encontrado, el valor del *hint* es el que se toma en lugar del que se encuentre en la definición, basado en el paso anterior.

Se pueden utilizar dos hints diferentes para especificar que un SELECT debe ser ejecutado en paralelo, estos son: PARALLEL y PARALLEL_INDEX. Si lo que se desea es que la instrucción SELECT no se ejecute en paralelo a pesar de las definiciones de las diferentes tablas involucradas se pueden utilizar los hints: NOPARALLEL y NOPARALLEL_INDEX.

El hint PARALLEL controla el grado de paralelismo que se aplica a las tablas a partir de un query, la sintaxis es la siguiente:

/+ PARALLEL (nombre_tabla [, grado [, instancias]]) */*

donde,

nombre_tabla Es el nombre de la *tabla* en donde se va a realizar el procesamiento en paralelo.

grado Especifica el número de procesos esclavos paralelos que cada *instancia*[2] va a contener cuando se ejecute la operación en esta *tabla*.

instancias Es el número de *instancias* que serán utilizadas por las operaciones paralelas en la *tabla*.

El alcance del hint paralelo es tan solo el query en el que está especificado. Los siguientes queries no son afectados por esta declaración.

El hint de PARALLEL_INDEX realiza lo mismo que el PARALLEL a diferencia que éste aplica solamente para índices. Puede ser utilizado para paralelizar rangos de búsqueda para índices particionados. La sintaxis de este hint es la siguiente:

/+ PARALLEL_INDEX (nombre_tabla [, índice [, grado [, instancias]]]) */*

donde,

nombre_tabla Es el nombre de la *tabla* en donde se va a realizar el procesamiento en paralelo.

índice Es el nombre del *índice* particionado en donde se paraleliza el rango del *índice* a examinar.

grado Especifica el número de procesos esclavos paralelos que cada *instancia* va a contener cuando se ejecute la operación en este *índice*.

Instancias Es el número de *instancias* que serán utilizadas por las operaciones paralelas en el *índice*.

Los valores para el *grado* e *instancias* pueden ser o números enteros o la palabra *DEFAULT*. La palabra *DEFAULT* indica que se utilice el valor predeterminado por la definición del objeto. *[MAH00]*

[2] Instancia. Es el medio por el cual se puede tener acceso a una base de datos Oracle. Técnicamente esta definida por una combinación de

DML paralelo.

El lenguaje de manipulación de datos (DML, del inglés Data Manipulation Language) paralelo utiliza mecanismos de ejecución paralela para acelerar o para realizar operaciones DML más complicadas sobre una gran base de datos.

Se puede paralelizar manualmente una operación DML utilizando múltiples sentencias DML simultáneamente para que afecten diferentes grupos de datos. Por ejemplo, uno puede paralelizar manualmente al:

- Utilizar múltiples sentencias *insert* en varias *instancias* de un *OPS* para hacer uso del espacio libre de las diferentes *instancias*.
- Utilizar múltiples sentencias *update* y *delete* con diferentes rangos de llaves o rangos de los identificadores de renglón que utiliza Oracle (rowid[3])

De cualquier forma, el paralelismo manual tiene las siguientes desventajas:

- Es difícil de utilizar, se tienen que abrir varias sesiones (posiblemente en diferentes *instancias*) y realizar múltiples sentencias.
- Falta de propiedades transaccionales, las sentencias *DML* son utilizadas en diferentes tiempos, por lo cual, los cambios son realizados sobre tomas diferentes de la *base de datos*. Para obtener una buena consistencia de datos se tienen que realzar sentencias commit[4] o *rollback*[5] para poder realizar dicha coordinación entre las *instancias*.
- División del trabajo compleja, se tienen que hacer consultas a los datos previas para poder obtener los rangos con los cuales se separarán las sentencias *DML*.

procesos de background y estructuras de memoria (SGA)
[3] Rowid. Es un identificador único de renglón en una base de datos Oracle.
[4] Commit. Es una instrucción de control a nivel transaccional, la cual guarda de forma permanente las modificaciones realizadas.
[5] RollBack. Esta instrucción deshace los cambios realizados en una transacción.

- Falta de afinidad y fuente de información, se necesita saber la afinidad de la información para poder ejecutar la sentencia *DML* correcta en la *instancia* correcta cuando se corre el *OPS*. También se tiene que averiguar como se encuentra la carga de trabajo para poder balancear el trabajo a través de las diferentes *instancias*.

Principalmente las operaciones *DML* paralelas son utilizadas para acelerar sentencias *DML* complicadas. El *DML* paralelo es muy útil en ambientes de sistemas de soporte para decisiones (DSS, del inglés Decision Support System) donde el rendimiento y el acceso a objetos grandes es muy importante. El *DML* paralelo complementa a las consultas paralelas al proveer las capacidades de actualizar y consultar una *base de datos* en un DSS.

La sobrecarga de la configuración del paralelismo hace que las operaciones del *DML* paralelo no sean una buena opción en sistemas OLTP[6]. Sin embargo, las operaciones *DML* paralelas pueden acelerar *tareas* batch[7] que corran sobre una *base de datos* en un ambiente OLTP.

En un sistema de datawarehouse[8], largas *tablas* necesitan ser actualizadas periódicamente con datos nuevos o modificados del sistema de producción. Este trabajo puede ser realizado eficientemente utilizando el *DML* paralelo.

Los datos que necesitan ser actualizados son generalmente cargados en una *tabla* temporal antes de realizar los procesos de actualización de datos. Esta *tabla* tiene renglones nuevos o renglones que han sido modificados desde la última actualización de la *base de datos* del data warehouse.

[6] OLTP. Del inglés Online Transaction Processing, es un tipo de proceso especialmente rápido en el que las solicitudes de los usuarios son resueltas de inmediato.
[7] Batch. Se dice que un proceso es batch cuando se realiza de forma secuencial y automática por la computadora.
[8] Datawarehouse. Es una tecnología y una disciplina orientada a la construcción de una colección de datos que permita y facilite el acceso a la información según lo requieran los procesos de toma de decisiones.

En un ambiente DSS, muchas aplicaciones requieren realizar operaciones complejas que requieren construir y manipular muchas *tablas* intermedias de resumen. A menudo las *tablas* de resumen son temporales y no son accesadas frecuentemente. El *DML* paralelo puede agilizar las operaciones sobre estas *tablas* intermedias.

Las sentencias *DML* pueden ser paralelizadas si y sólo si se ha activado esta propiedad en la sesión por medio de la opción *ENABLE PARALLEL* DML a través de la sentencia *ALTER SESSION*. Esta habilitación es requerida ya que la forma de utilizar los candados, las transacciones y requerimientos de espacio en disco son diferentes cuando se utiliza el *DML* paralelo y el *DML* serial.

Las sesiones tienen desactivada la opción de *DML* paralelo por *default*. Cuando el *DML* paralelo está desactivado, ninguna sentencia *DML* será ejecutada en paralelo aún si tiene declarado el *hint PARALLEL*.

Cuando el *DML* paralelo está activado en una sesión, todas las sentencias *DML* son consideradas para ejecutarse en paralelo. Sin embargo, aún cuando el *DML* paralelo está activado, las operaciones *DML* seguirán ejecutándose serialmente si no es especificado ningún *hint* o cláusula o si algunas de las restricciones de ejecución en paralelo es violada. *[LEV99]*

Transacciones con DML en paralelo.

Si se tiene en la sesión activado el *DML* paralelo se debe tener un cuidado especial en las transacciones, si cualquier sentencia *DML* dentro de una *transacción* modifica una *tabla* en forma paralela, ninguna sentencia *DML* podrá accesar a dicha *tabla*, ya sea paralela o serial, durante el resto de la *transacción*. Esto quiere decir, que las modificaciones en paralelo realizadas en una *transacción* no se pueden ver mientras se ejecuta la misma.

Las sentencias paralelas o seriales que intenten acceder a una *tabla* que ha sido modificada en paralelo durante la misma *transacción* serán rechazadas y causarán un error.

Para ejecutar una operación *DML* en paralelo, el coordinador de ejecución en paralelo produce ejecuciones en paralelo en diversos servidores y cada servidor de ejecución en paralelo realiza una porción del trabajo dentro de su propia *transacción* de procesos paralelos.

- Cada servidor de ejecución en paralelo crea una diferente *transacción* de procesos paralelos.
- Para reducir conflictos en los *segmentos de rollback*[9], solamente unas pocas transacciones con procesos paralelos deben de encontrarse en el mismo *segmento de rollback*.

El coordinador también tiene su propio coordinador de transacciones, el cual puede tener su propio *segmento de rollback*.

Oracle asigna a las transacciones los *segmentos de rollback* que tienen un menor número de transacciones activas. Para poder agilizar los procesos se deben crear y activar suficientes *segmentos de rollback* para que así a lo más queden asignadas dos transacciones con procesos paralelos a un *segmento de rollback*.

Es necesario que los *segmentos de rollback* sean creados en *tablespaces* que tienen suficiente espacio en disco para que éstos puedan crecer cuando sea necesario y asignar el valor de UNLIMITED al parámetro MAXEXTENTS para el almacenamiento de los *segmentos de rollback*. *[MAH00]*

[9] Segmentos de rollback. Son estructuras utilizadas en las bases de datos Oracle, en las cuales se guarda la información necesaria para poder deshacer una transacción en caso de que ocurra un rollback.

Restricciones con DML en paralelo.

A continuación se mencionarán las restricciones que se deben tomar en cuenta cuando se utilice el *DML* paralelo:

- Las operaciones *UPDATE* y *DELETE* no se pueden paralelizar en *tablas* no particionadas.
- Una *transacción* puede tener varias sentencias *DML* paralelas que modifiquen diferentes *tablas*, pero una vez realizada la modificación ninguna consulta o modificación podrá accesar a la *tabla.*
- Si se tiene el parámetro de inicialización ROW_LOCKING = INTENT, entonces las sentencias *UPDATE, DELETE* e *INSERT* no serán paralelizadas.
- Los Triggers[10] no soportan operaciones de *DML* paralelo.
- La funcionalidad de replicación tampoco se puede paralelizar sentencias *DML*.
- El *DML* paralelo no se puede realizar en paralelo si se cuenta con alguno de los siguientes *constraints*[11]: de integridad referencial propia, borrado en cascada, integridad deferenciada.
- El *DML* paralelo no se puede ejecutar en *tablas* que contengan objetos como columnas o de tipo *LOB*.
- Una *transacción* que contenga una operación de *DML* paralelo no puede ser o convertirse en una *transacción* distribuida.

Si ocurre alguna violación de estas restricciones la sentencia será realizada en forma serial sin presentar algún tipo de advertencia o error, a menos que se trate de accesar a una *tabla* que ya ha sido modificada por un proceso paralelo, esta es la única restricción con la cual si se obtiene un error. *[MAH00]*

[10] Trigger. Es un bloque de código que es ejecutado cuando ocurre cierto evento de la base de datos. Hay tres tipos de eventos en una base de datos: insert, update y delete.

DDL paralelo.

Se pueden paralelizar operaciones de *Data Definition Languaje* o *DDL* para *tablas* e *índices* que estén particionados o no. Las operaciones *DDL* para *tablas* o *índices* no particionados son:

- *CREATE INDEX*
- *CREATE TABLE ... AS SELECT*
- *ALTER INDEX ... REBUILD*

Las operaciones *DDL* para *tablas* o *índices* particionados son:

- *CREATE INDEX*
- *CREATE TABLE ... AS SELECT*
- *ALTER TABLE ... MOVE PARTITION*
- *ALTER TABLE ...SPLIT PARTITION*
- *ALTER TABLE ...COALESCE PARTITION*
- *ALTER INDEX ... REBUILD PARTITION*
- *ALTER INDEX ... SPLIT PARTITION*

La instrucción CREATE TABLE para una *tabla* de ordenación indexada puede ser paralelizada con o sin la sentencia SELECT as.

Las operaciones *DDL* no se pueden ejecutar en paralelo para *tablas* que contengan columnas de tipo *LOB* u objeto.

[11] Constraint. Un constraint forza ciertos aspectos de integridad de datos en unas base de datos. Cuando se agrega un constraint a una columna en particular, Oracle automáticamente asegura que de ninguna manera sea aceptado un dato el cual no cumpla con lo establecido en dicho constraint.

Cuando se crea una *tabla* o un *índice* usando una instrucción en paralelo, dos o más procesos esclavos trabajan en una parte de la instrucción para poder crear el objeto; cada proceso esclavo crea un segmento[12] temporal durante el proceso de creación, al final, el coordinador de procesos une cada uno de estos segmentos liberando el espacio no ocupado y lo convierte en un solo segmento final.

Cuando se utiliza *DDL* paralelo para crear objetos tales como *tablas* e *índices*, se necesita poner atención a dos puntos fundamentales:

- Como es alojado el espacio.
- El potencial de la fragmentación.

Uno debe ser muy cuidadoso mientras especifica el lugar de almacenamiento de las *tablas* o *índices* que sean creados en paralelo, porque la cantidad de espacio a alojar puede ser mucho mayor de lo que se puede esperar. Cuando se crea una *tabla* o un *índice* utilizando *DDL* paralelo, cada proceso esclavo reserva espacio basado en su especificación de almacenamiento. Por ejemplo, si se crea una *tabla* con un INITIAL 10 MB y un grado de paralelismo de 4, entonces cada proceso esclavo reserva 10 MB de memoria para él; como resultado tenemos que se utilizará un total de 40 MB durante la creación de dicha *tabla*.

Por el lado de la fragmentación tenemos que cada segmento va a tener un espacio libre el cual será liberado al finalizar la creación del objeto y estos espacios se le regresarán a la *base de datos* para que los utilice cuando ella lo necesite; por lo tanto este espacio libre queda entre los diferentes segmentos provocando que el objeto se encuentre fragmentado. *[LEV99]*

[12] Segmento. Es un objeto lógico que ocupa espacio en una base de datos Oracle, como una tabla o un índice.

III Oracle Parallel Server (OPS)

La herramienta utilizada en este trabajo es Oracle Parallel Server, por lo cual se desglosará su arquitectura y analizarán cada una de sus partes como las funciones de éstas. También se comprenderá la funcionalidad de la herramienta para poder entender la forma en que son ejecutadas las instrucciones y su relación con la configuración del ambiente de trabajo.

¿Qué es Oracle Parallel Server?

Es una herramienta la cual permite a varias instancias en diferentes equipos acceder simultáneamente a una misma base de datos. Mientras las características de la ejecución en paralelo permiten reflejar el poder de dos o más procesadores en un sistema, Oracle Parallel Server (*OPS*) permite reflejar el poder de varios sistemas en una sola *base de datos*. Estas dos características pueden ser combinadas, aprovechando la combinación del poder de procesamiento de varios procesadores en varias computadoras, para manejar una gran cantidad de usuarios o para realizar trabajos largos y complicados. No sólo se pueden agregar más procesadores para aumentar el soporte, también se pueden agregar equipos completos.

En la figura 3.1 se muestra una *base de datos* compuesta por dos nodos. Cada nodo corre una *instancia* de la *base de datos*, cada *instancia* tiene su grupo de procesos background y su propia *SGA* (*System Global Area*). Ambas *instancias* montan y abren una *base de datos* que se encuentra en un subsistema de discos compartidos.

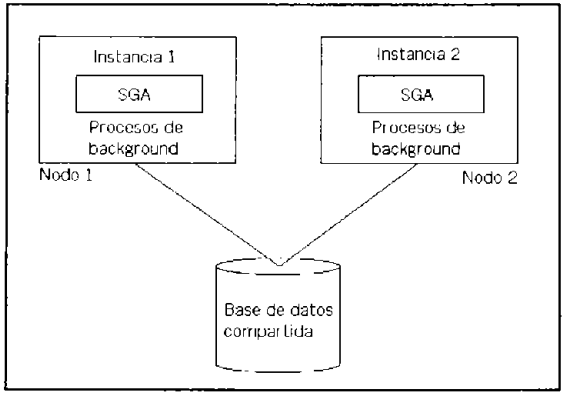


Figura 3.1 *OPS* permite que varias *instancias* operen sobre una sola *base de datos*.

Un sistema paralelo de *base de datos* tiene una configuración relacionada a varias *instancias*, por lo cual este sistema tiene algunas ventajas con respecto a uno con una sola *instancia* o standalone. Estas ventajas son:

- Alta disponibilidad
- Mejor escalabilidad
- Balanceo de carga

Cuando una *instancia* se cae, las otras *instancias* siguen funcionando. Esto hace que la disponibilidad aumente, porque la falla de una *instancia* no da como consecuencia que la *base de datos* no esté disponible para los usuarios. Los usuarios conectados a otras *instancias* siguen trabajando sin ningún contratiempo. Por otro lado, los usuarios conectados a la *instancia* con problemas pueden reconectarse a cualquier otra *instancia* que esté funcionando correctamente.

Si por necesidades del sistema se cuentan con más usuarios de los que se pueden controlar con los nodos existentes, fácilmente se puede agregar otro nodo y dar de alta una nueva *instancia* en éste. Esto es lo que permite una mejor escalabilidad de la con que se contaría en un sistema con una sola *instancia*. Oracle cuenta con otras características que permiten el balancear el trabajo entre las diferentes *instancias*, permitiendo optimizar la carga de cada nodo.

Administrar una *base de datos* con *OPS* es más complicado, ya que no sólo se tiene el problema de manejar varias *instancias*, también se tienen que tomar en cuenta varios puntos para poder obtener un rendimiento óptimo del sistema los cuales se explicarán a lo largo de este capítulo. *[MAH00]*

Arquitectura del Oracle Parallel Server

Antes de que se pueda instalar y configurar satisfactoriamente *OPS*, se tienen que estudiar varios componentes de la arquitectura como son: la cantidad de *instancias* que se van a manejar, la forma en que se va a realizar la comunicación entre estas *instancias*, la forma en que se deben configurar los data files, *control files* y otros. También se tiene que hacer un análisis de los *segmentos de rollback* que se van a utilizar.

Todos estos puntos son de los que se va a hablar a continuación para poder instalar y configurar la herramienta de *OPS* y aprovechar de la mejor forma posible sus ventajas.

Instancias y OPS

Una de las grandes diferencias entre una *base de datos* en *OPS* y una standalone, es que con *OPS* se cuenta con varias *instancias* que operan con la *base de datos* al mismo tiempo.

Con *OPS*, una *base de datos* es montada y abierta por varias *instancias* concurrentemente. Cada Instancia en *OPS* es como cualquier instancia de Oracle que está compuesta por su *SGA* y varios procesos de background. Cada *instancia* corre en un equipo distinto por lo cual cada una tiene su propia memoria y CPU (Central Process Unit). Cada nodo puede tener diferente arquitectura o la misma como sería una SMP. La cantidad de memoria y el número de procesadores no tienen que ser los mismos en los diferentes nodos.

La ubicación de la *base de datos* se encuentra en un subsistema de disco los cuales tienen que estar compartidos o accesibles por todos los nodos. En la figura 3.2 se muestra un sistema de dos nodos.

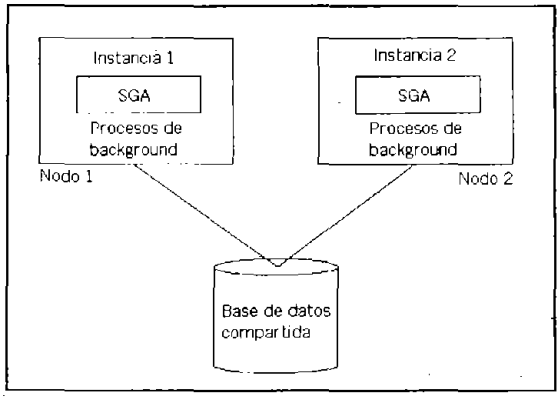


Figura 3.2 Una base de datos *OPS* con dos nodos.

Características de las instancias *OPS*

Al trabajar con una arquitectura de varias *instancias*, la configuración del *OPS* tiene algunas características muy especiales:

- Los archivos data files y de control son compartidos por todas las *instancias*.
- Cada *instancia* tiene su grupo de archivos *redo logs*.
- Un ambiente *OPS* cuenta con el IDLM (Integrated Distributed Lock Manager) para controlar la sincronización entre *instancias*. El IDLM consta de varios procesos background.
- Un ambiente *OPS* también cuenta con un GMS (Group Membership Service) para poderse comunicar con el administrador de clusters del sistema operativo, es un componente el cual coordina las actividades de varios nodos en un cluster.
- Cada *instancia* tiene su propio *segmento del rollback*.

Requerimientos del *OPS*

Si se va a utilizar *OPS* en un equipo primero se tienen que tomar en cuenta los siguientes requerimientos:

- La plataforma del hardware y sistema operativo deben de soportar una arquitectura de discos compartidos.
- El software de sistema operativo debe soportar "clustering", y debe tener un componente para administrar nodos que se encuentren en el ambiente del *OPS*.
- Cada nodo en el cluster debe tener su propio directorio de ORACLE_HOME con su copia de Oracle Server. En todos los nodos se debe de contar con la misma versión de Oracle.
- Para sistemas con equipos Unix, todos los archivos de la *base de datos* (data files, *control files* y *redo log*) deben de estar en un dispositivo "raw".

Comparación de configuraciones entre *OPS* y standalone.

Instancia Standalone	**Oracle Parallel Server**
Una *base de datos* es montada por una *instancia*.	Una *base de datos* es montada en modo compartido por múltiples *instancias*.
Tiene una *SGA* (*System Global Area*) y un grupo de procesos background.	Tiene múltiples *SGA* y múltiples grupos de procesos background, un grupo de procesos por cada nodo. Son requeridos procesos background adicionales. También se necesita de un coordinador entre múltiples *instancias* llamado IDLM.

Todos los bloqueos son realizados a nivel *instancia*.	Los bloqueos globales deben ser administrados en varias *instancias*.
No hay una sincronización elevada.	Si existe una elevada sincronización.
Corre en equipos con un solo procesador o en un equipo SMP.	Corre en equipos *MPP* o cluster.
	Son requeridos parámetros globales constantes.
Vistas V$ para monitorear la *instancia*.	Vistas V$ para monitorear la *instancia* local.
	Vistas GV$ para monitorear todas las *instancias*.
Compatible con casi todo tipo de aplicaciones.	No es compatible con todo tipo de aplicaciones por el alto requerimiento de sincronización.
Disponibilidad limitada a una *instancia*.	Al contar con múltiples *instancias*, se tiene una alta disponibilidad.
La escalabilidad está limitada a una *instancia*.	La escalabilidad se extiende añadiendo más *instancias*.

Sincronización entre instancias.

En la configuración del *OPS*, múltiples *instancias* montan y abren una *base de datos* en modo compartido. Al tener varias *instancias* compartiendo una *base de datos*, es como si varias copias del mismo objeto se pusieran en el *SGA* de más de una *instancia*. Por ejemplo, cuando la *instancia* 1 necesita modificar un renglón que se encuentra en el bloque B1, ésta lee del disco el bloque B1 y lo carga en su *buffer* de caché y entonces modifica el bloque. Ahora, si la *instancia* 2 necesita modificar el bloque B1, necesita una copia del bloque B1 en su *buffer* de caché. Para que la *instancia* 2 obtenga la última versión del bloque B1, la *instancia* 1 necesita escribir el bloque en disco y entonces la *instancia* 2 puede leer y proceder a modificar el bloque. Este proceso se ilustra en la Figura 3.3.

Porque dos *instancias* pueden modificar la copia de un bloque que resida en su *buffer* de caché respectivo, es necesario que se mantenga la consistencia entre estas dos copias. En otras palabras, retomando el ejemplo anterior, la *instancia* 2 necesita de alguna manera saber que la *instancia* 1 ha leído y modificado el mismo bloque que se va a utilizar. El mantener la consistencia entre las versiones almacenadas en los *buffers* de caché de los bloques de la *base de datos* en varias *instancias* es llamado *coherencia caché*. Oracle utiliza bloqueos globales para garantizar la coherencia caché, estos bloqueos son globales ya que son administrados a través de todas las *instancias* asociadas con la *base de datos* del *OPS*. El componente de Oracle encargado de coordinar estas actividades se llama Integrated Distributed Lock Manager (IDLM).

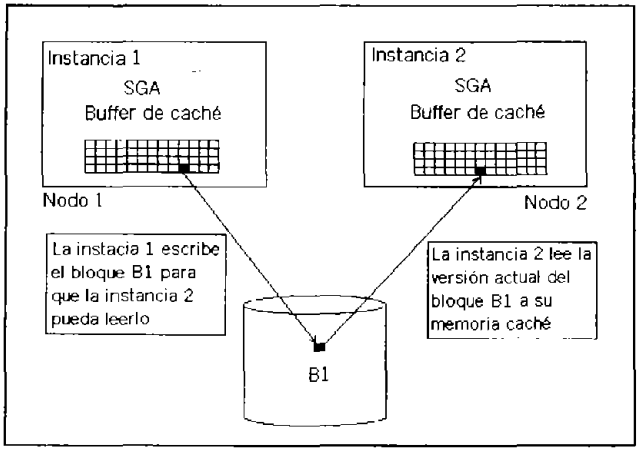


Figura 3.3 Coherencia de caché.

Integrated Distributed Lock Manager (IDML)

Los recursos de la *base de datos* como bloques de datos, bloques de *índices*, *segmentos de rollback* y el *diccionario de datos* son compartidos por todas las *instancias* del *OPS*. Para prevenir conflictos, se necesita un mecanismo de bloqueo para controlar el acceso a dichos recursos. El IDLM realiza este tipo de bloqueos y hace lo siguiente:

- Mantiene una lista de los recursos compartidos de la *base de datos*.
- Aloja los bloqueos en estos recursos.
- Coordina los requerimientos de bloqueo.
- Mantiene el rastro de los bloqueos concedidos en los recursos.

El IDLM está integrado por dos procesos background, el Lock Manager Daemon (LMD) y el Lock Monitor (LMON). Estos componentes son ilustrados en la Figura 3.4

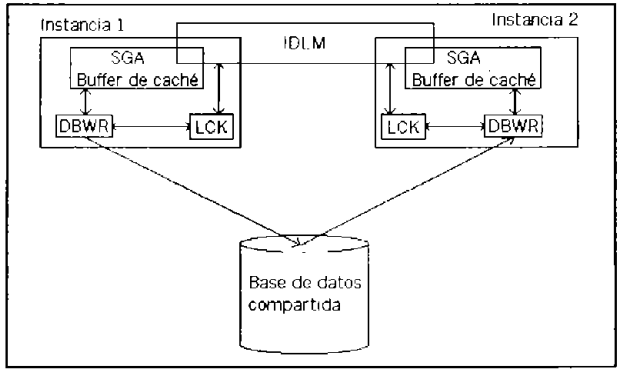


Figura 3.4 Componentes del IDLM

El estado de bloqueo de varios recursos de la *base de datos* es mantenida en un área de la *SGA* conocida como Area de Bloqueo Distribuida. Esta información de bloqueo está distribuida a través de todos los nodos del *OPS*. Cada nodo mantiene el estatus de bloqueos de un subconjunto de los recursos de la *base de datos*. El nodo que mantiene la información de un recurso en particular de la *base de datos* es llamado nodo maestro para el recurso en particular. Cuando un bloqueo es requerido por algún recurso de un nodo es dirigido al nodo maestro de ese recurso. Si un nodo falla, solo un subconjunto de la información del bloqueo necesita ser distribuida por toda la red. Similarmente, cuando las *instancias* son inicializadas o detenidas, la información de bloqueo es redistribuida a través de todos los nodos activos.

El IDLM es una de las grandes diferencias entre las versiones 7 y 8. El subsistema de administración de bloqueo utilizado en la versión 7 es externo a Oracle y es conocido como Distributed Lock Manager (DLM). Históricamente el DLM ha sido suministrado por el vendedor del sistema operativo como un componente del mismo. A consecuencia de que cada fabricante de sistema operativo proveía de su propia implementación de DLM, la funcionalidad tendió a ser diferente en cada plataforma. Lógicamente esto se convirtió en un problema, por eso con la salida de la versión 8, la funcionalidad del administrador de bloqueo ha sido integrada en el software del Oracle. El IDLM de Oracle8 provee consistencia a través de las diferentes plataformas.

Segmentos de rollback en OPS

Todas las *instancias* en un *OPS* necesitan *segmentos de rollback* para almacenar la información que se puede deshacer. El *segmento de rollback* de SYSTEM es compartido y utilizado por todas las *instancias* para transacciones que contienen objetos de sistema. Adicionalmente al *segmento de rollback* de SYSTEM, cada *instancia* necesita de por lo menos un *segmento de rollback* para ella misma. No podrá ser posible inicializar una *instancia* y abrir la *base de datos* a menos que la *instancia* pueda obtener por lo menos un *segmento de rollback*. Si la *instancia* no puede adjudicarse aunque sea un *segmento de rollback*, la *instancia* se va a inicializar y va a montar la *base de datos*, pero no la va a abrir. Un *segmento de rollback* sólo puede ser adjudicado por una sola *instancia*, por lo tanto, uno debe tener por lo menos tantos *segmentos de rollback* en la *base de datos*, como número de *instancias* concurrentes que abran la *base de datos*.

Un *segmento de rollback* asignado a una *instancia* permanece exclusivamente para esta *instancia*, ninguna otra *instancia* puede escribir información para deshacer en este *segmento de rollback*. Sin embargo, todas las *instancias* pueden leer la información de este *segmento de rollback* para poder mantener la consistencia de lectura.

Los *segmentos de rollback* pueden ser públicos o privados. Un *segmento de rollback* público es creado utilizando la opción *PUBLIC* dentro del comando *CREATE ROLLBACK SEGMENT*. Cualquier *segmento de rollback* creado sin la opción de *PUBLIC* es considerado privado. Un *segmento de rollback* público puede ser utilizado por cualquier *instancia*, en cambio, un *segmento de rollback* privado puede ser utilizado por cualquier *instancia* que sea declarada dentro del parámetro de inicialización llamada *ROLLBACK_SEGMENTS* de dicha *instancia*.

Los *segmentos de rollback* privados son preferidos para una administración sencilla. Si todos los *segmentos de rollback* son privados uno puede saber que *segmento de rollback* va a ser utilizado por cada *instancia*, lo cual no es verdad cuando los segmentos son públicos. Otra ventaja de los *segmentos de rollback* privados es que se puede colocar los *segmentos de rollback* utilizados por cada *instancia* en *tablespaces* separados, y estos *tablespaces* pueden ser distribuidos por los discos para poder evitar cuellos de botella en la lectura de dichos discos.

El parámetro *ROLLBACK_SEGMENTS*, del archivo de inicialización, contiene una lista de todos los nombres de los *segmentos de rollback* que una *instancia* utilizará. Siempre que se inicializa una *instancia*, intentará obtener todos los *segmentos de rollback* especificados por este parámetro, si existe algún problema al obtener aunque sea solo un segmento la *instancia* no podrá iniciar y resultará un error. El valor por defecto de dicho parámetro es *NULL*, lo cual significa que la *instancia* intentará obtener cualquier *segmento de rollback* público que no sea utilizado por alguna otra *instancia*; por lo cual se recomienda especificar los *segmentos de rollback* privados a utilizar por cada *instancia*, como se muestra a continuación:

ROLLBACK_SEGMENTS = (RBS_INSTANCIA_A1, RBS_INSTANCIA_A2)

Una *instancia* necesita de por lo menos un *segmento de rollback* ya sea público o privado para poder abrir la *base de datos*, de cualquier modo, normalmente o deseablemente se requiere que una *instancia* tenga más de un *segmento de rollback*. Son dos los parámetros de inicialización

que nos indican cuantos *segmentos de rollback* se necesitan por *instancia*: *TRANSACTIONS* y *TRANSACTIONS_PER_ROLLBACK*. Se utiliza la siguiente fórmula para calcular dicho número:

ROLLBACK_SEGMENTS = TRANSACTIONS / TRANSACTIONS_PER_ROLLBACK

Ejecución paralela en Oracle Parallel Server

Cómo funciona la ejecución en paralelo con OPS

En un ambiente de ejecución en paralelo, Oracle divide una sentencia SQL en *tareas* mas pequeñas y asigna estas *tareas* a múltiples procesos esclavos. Entonces los procesos esclavos ejecutan sus *tareas* concurrentemente. *OPS* extiende esta característica al ejecutar procesos esclavos en paralelo en múltiples nodos del servidor paralelo.

Esto es, si se ejecuta una consulta en un servidor paralelo con cuatro nodos, éste las divide y las ejecuta por separado y luego junta los resultados obtenidos por los procesos esclavos, como se muestra el siguiente ejemplo:

```
SELECT /*+ PARALLEL (ordenes, 2, 2) */
Count(*)
FROM ordenes;
```

Esta sentencia contiene el *hint* PARALLEL la cual especifica que se ejecute con un grado 2 y en 2 *instancias*. Lo que haría Oracle es dividir el trabajo para que se ejecutara en dos *instancias* utilizando dos procesos esclavos en cada una para realizar la consulta, como se muestra en la Figura 3.5.

La especificación del grado de paralelismo consiste en dos partes: el grado y las *instancias*. La parte de *instancias* controla el número de *instancias* que se utilizaran para ejecutar la sentencia en paralelo. El grado especifica el número de procesos esclavos paralelos que trabajaran en cada *instancia*.

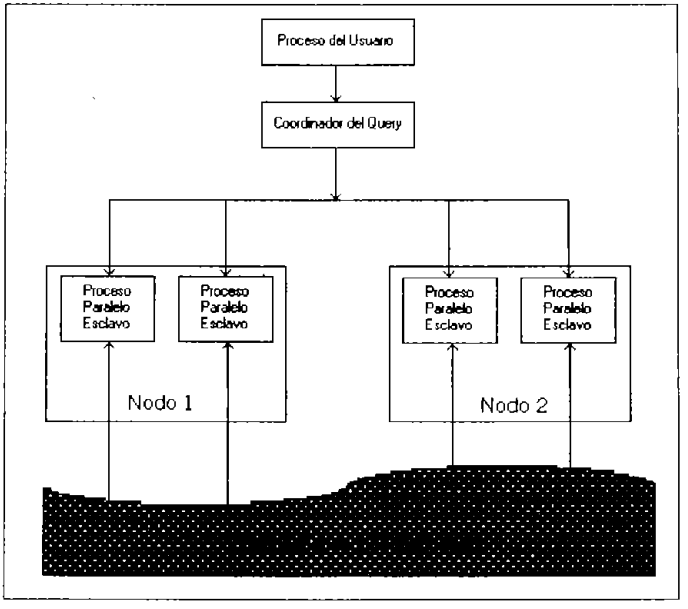


Figura 3.5 Dos *instancias* ejecutando una misma consulta en paralelo.

Afinidad de discos.

Se dice que una *instancia* de *OPS* tiene afinidad por un recurso, si el recurso es directamente accesible desde el nodo donde se está corriendo la *instancia*. En la figura 3.6 se muestran dos *instancias* en dos nodos distintos: el disco A está conectado directamente al nodo 1, y el disco B está conectado directamente al nodo 2. Una red de alta velocidad compone la arquitectura de discos compartidos en donde los dos discos son accesibles desde cualquier nodo. Como el disco A es local para el nodo 1, se dice que la *instancia* corriendo en el nodo 1 tiene

afinidad por el disco A, así como la *instancia* 2 tiene afinidad por el disco B. Las operaciones de E/S se ejecutarán con mayor eficiencia cuando la *instancia* acceda a los discos con los cuales tiene afinidad.

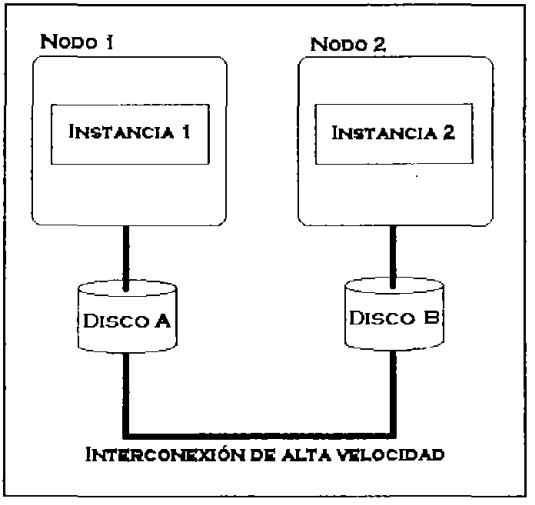


Figura 3.6 Afinidad de discos.

Cuando se descomponen las *tareas* de ejecución en paralelo en procesos esclavos en múltiples nodos, Oracle toma en cuenta la afinidad. De cualquier forma la afinidad es transparente para las aplicaciones o usuarios realizando estas *tareas*.

Cuando la afinidad en discos no es utilizada, Oracle balancea el alojamiento de los procesos esclavos paralelos a través de las *instancias* disponibles. Cuando la afinidad de discos es utilizada, Oracle intenta alojar a los procesos esclavos paralelos en la instacia que esta lo mas cercano posible a los datos que requieran estos procesos. la afinidad de discos puede reducir considerablemente las comunicación entre nodos y mejorar el desempeño de las operaciones paralelas.

Aún cuando la afinidad de discos es soportada, no es utilizada por todas las operaciones paralelas, en la tabla 3.1 se muestran las operaciones con las cuales Oracle utilizará la afinidad de discos y en cuales no.

Tabla 3.1 utilización de la afinidad en operaciones paralelas.

Operación paralela	**Afinidad de discos utilizada**
Busqueda en una *tabla*	Sí
Alojamiento de un *tablespace* temporal	Sí
Insert, *Update* o *Delete*	Sí
Busqueda en un *índice*	Sí
Creación de *tabla*	No
Creación de *índice*	No

IV Implementación y resultados del OPS

En este capítulo se van a enseñar los pasos que se siguieron para poder hacer una prueba con Oracle Parallel Server sobre una arquitectura SMP. Se muestran los pasos que se siguieron para hacer una migración a OPS, como se debe de iniciar y trabajar con él. Se hace referencia a los resultados obtenidos de distintas consultas y se ve la diferencia con la arquitectura anterior.

Iniciando una base de datos con OPS

Para crear una *base de datos* con *OPS*, se realiza de la misma manera que una tradicional o standalone, lo que cambia son los parámetros del archivo de inicialización de la *base de datos*. La forma de iniciar la *instancia* y el comando *CREATE DATABASE* es exactamente lo mismo para los dos tipo de *base de datos*. En realidad lo que se hace es crear la *base de datos* y luego transformarla en una *base de datos OPS*.

Para poder crear una *base de datos OPS* se debe de hacer lo siguiente:

- Asegurarse de que se cuenta con la configuración de hardware y software necesarios.
- Organizar el disco duro del equipo para alojar el software de Oracle, los archivos de la *base de datos* y otros archivos relacionados.
- Instalar el software de Oracle, incluyendo la opción de Oracle Parallel Server, en todos los nodos donde será utilizado.
- Crear un archivo de parámetros de inicialización para cada una de las *instancias*. Se recomienda crear un archivo de parámetros globales compartido para todas las *instancias*.
- Crear la *base de datos* mediante el comando *CREATE DATABASE*. *[BAM99]*

Cuando se escriba el comando *CREATE DATABASE* para la creación de una nueva *base de datos OPS*, se necesitará tener consideraciones especiales con las siguientes opciones:

- *MAXINSTANCES*. Especifica el número máximo de *instancias* que pueden acceder a la *base de datos* concurrentemente. El valor por default es 2 y si se va a utilizar *OPS* es muy probable que se tengan más de dos *instancias* por lo que este valor sería

inadecuado. Se recomienda que este valor sea mayor al número de *instancias* planeadas para poder permitir un crecimiento futuro.

- *MAXLOGFILES*. Es el número máximo de grupos de *redo log* que pueden llegar a ser creados en la *base de datos*. Cada *instancia* en un ambiente *OPS* tiene sus propios grupos de *redo log*. Comúnmente, a los grupos de *redo log* se les conoce como un *thread,* por consiguiente cada *instancia* tiene su propio *thread* de *redo log*. Para obtener el valor de *MAXLOGFILES* se debe de multiplicar el número esperado de *threads* de *redo log* por el número esperado de grupos de *redo log* en cada *thread*.
- *MAXLOGHISTORY*. Señala el número máximo de archivos *redo log* que pueden ser almacenados en el log histórico del archivo de control para poder realizar una recuperación automática. El valor por default es cero, el cual inactiva esta opción. Cuando se alcanza el número especificado en *MAXLOGHISTORY* la historia más antigua es reemplazada por la actual.
- *MAXDATAFILES*. Define el número máximo de *datafiles* que pueden ser creados para la *base de datos*. Generalmente, una *base de datos OPS* tiene más *datafiles* que una standalone, por esto se debe de especificar un valor respectivamente mayor. *[ERN01]*

Si no se toman las precauciones necesarias con estas opciones durante el proceso de creación de la *base de datos*, sólo podrán ser cambiadas recreando el archivo de control de la *base de datos*.

Los parámetros de inicialización se pueden dividir en tres categorías:

- Algunos parámetros **deben ser iguales** en todas las *instancias*. Estos parámetros son conocidos como parámetros globales constantes. Para realizar algún cambio sobre estos parámetros se necesita detener todas las *instancias*, hacer el cambio y reiniciar.
- Otros parámetros **pueden ser diferentes** para cada *instancia*.

- Algunos parámetros **deben ser diferentes** en cada *instancia*.

La administración de estos tres tipos de parámetros debe permitir un fácil acceso para modificarlos en caso de que sea requerido y así evitar al máximo el detener todas las *instancias*.

En la tabla 4.1 se muestran divididos los parámetros utilizados para utilizar una configuración *OPS*. *[STE99]*

Parámetros globales constantes (Deben ser iguales)	**Preferentemente iguales (Pueden ser diferentes)**	**Deben ser diferentes**
PARALLEL_SERVER	PARALLEL_MAX_SERVERS	ROLLBACK_SEGMENTS
PARALLEL_SERVER_INSTANCES	PARALLEL_DEFAULT_ MAX_INSTANCES	THREAD
DB_BLOCK_SIZE	LM_LOCKS	INTANCE_NUMBER
DB_NAME	LM_PROCS	
CONTROL_FILES	LM_RESS	
GC_FILES_TO_LOCKS	LOG_ARCHIVE_FORMAT	
GC_LCK_PROCS		
GC_ROLLBACK_LOCKS		
CACHE_SIZE_THRESHOLD		
COMPATIBLE		
CONPATIBLE_NO_RECOVERY		
DB_DOMAIN		
DB_FILES		
DML_LOCKS		
FREEZE_DB_FOR_FAST_ INSTANCE_RECOVERY		
LOG_FILES		

MAX_COMMIT_PROPAGATION_ DELAY		
ROW_LOCKING		

Debido a que los parámetros de las *instancias* no van a ser iguales en todas ellas, cada *instancia* debe tener su propio archivo de parámetros. Se recomienda utilizar dos archivos de parámetros, uno que contenga los parámetros utilizados por todas las *instancias* y que son iguales y otro con los parámetros que son diferentes.

Cuando se utilice un archivo con los parámetros globales se tiene que emplear la directiva *IFILE* para incluir este archivo al iniciar la *instancia*. Por ejemplo, si tenemos una *base de datos* en *OPS* llamada PRUEBA con dos *instancias* llamadas PRUEBA1 y PRUEBA2 respectivamente; estas dos *instancias* se encuentran en dos nodos separados. Cada *instancia* tiene su propio archivo de parámetros, almacenado en cada nodo. Los nombres de los archivos de parámetros son $ORACLE_HOME/dbs/initPRUEBA1.ora y $ORACLE_HOME/dbs/initPRUEBA2.ora respectivamente; sin embargo, el archivo con los parámetros globales se encuentra en el archivo init_comun.ora en un directorio accesible por los dos nodos al cual llamaremos $COMUN. Para incluir el archivo con parámetros globales se tiene que añadir la siguiente línea en los archivos de parámetros de cada *instancia*:

IFILE = $COMUN/initComun.ora

Al momento de iniciar cada *instancia*, Oracle lee el archivo de parámetros específico de la *instancia* y cuando encuentra la directiva *IFILE* lee el archivo común de parámetros especificado, como se muestra en la figura 4.1*[MAH00]*

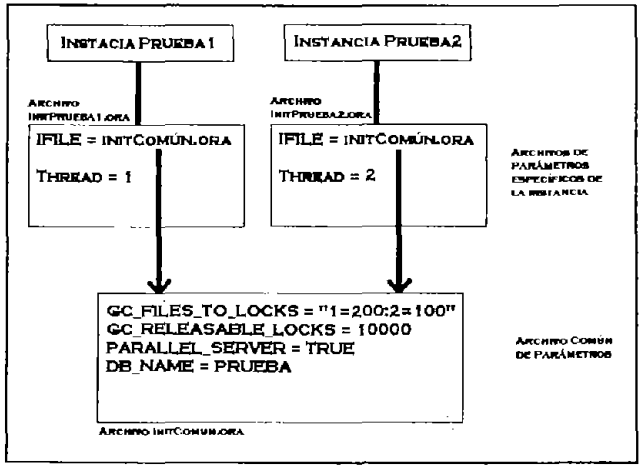


Figura 4.1 Dos *instancias* compartiendo un archivo común de parámetros.

Una *base de datos OPS* es creada con la opción de *parallel server* desactivada. Sólo después de que ésta es creada se habilita el *OPS* y se abre la *base de datos* como *OPS*. Se requiere hacer los siguientes pasos en uno de los nodos para crear la *base de datos OPS*:

1. Desactivar la opción de *OPS* en el archivo de inicialización (*PARALLEL_SERVER=FALSE*).
2. Para Oracle8, inicializar el grupo de servicios de miembros (group membership services o GMS) en el nodo donde se creó la *base de datos*. Este paso no aplica para Oracle8i.
3. Inicializar el server manager y entrar como internal (*CONNECT INTERNAL*).
4. Inicializar la *instancia* en modo *NOMOUNT*.
5. Ejecutar el comando *CREATE DATABASE*.

Administrando una base de datos con OPS

Inicializar una *base de datos OPS* es un poco diferente a inicializar una *base de datos* standalone. Una *instancia* de una *base de datos OPS* se puede inicializar de dos maneras

distintas: de modo exclusivo o de modo compartido. De modo exclusivo, solamente una *instancia* puede montar y abrir la *base de datos* y de modo compartido, múltiples instancias pueden montar y abrir la *base de datos* simultáneamente. El parámetro de inicialización que determina el modo en el que se va a trabajar con la *base de datos* es *PARALLEL_SERVER*. El modo compartido es el estado normal utilizado en una *base de datos OPS*, y el modo exclusivo es utilizado para realizar ciertas *tareas* administrativas como por ejemplo cambiar de el modo de *ARCHIVELOG* de ON a OFF.

Una *instancia* puede ser inicializada en dos modos: exclusivo o compartido.

- **Modo Exclusivo**. Para iniciar una *base de datos* en modo exclusivo, se debe de escoger una *instancia* para trabajar, y apagar el *OPS* para esa *instancia* modificando el parámetro PARALLEL_SERVER con el valor FALSE en el archivo de inicialización. Solamente una *instancia* puede abrir la *base de datos* de modo exclusivo, por lo cual, primero se deben de detener todas las *instancias* que normalmente abren la *base de datos* de modo compartido.
- **Modo Compartido**. Si el parámetro PARALLEL_SERVER tiene el valor de TRUE, se pueden iniciar varias *instancias* de modo compartido al mismo tiempo. Para lograr esto se debe de tener por lo menos un *segmento de rollback* y también por lo menos un thread de *redo log* con por lo menos dos grupos de *redo log*.

Las *instancias* se pueden agrupar de manera lógica, esto se realiza mediante el parámetro INSTANCE_GROUPS del archivo de inicialización; dicho parámetro define grupos de *instancias* y asigna las *instancias* a estos. Por ejemplo, si se tiene la siguiente línea en el archivo de parámetros, quiere decir que la *instancia* pertenece al grupo A:

INSTANCE_GROUPS = grupoA

Una *instancia* puede pertenecer a varios grupos de *instancias* al mismo tiempo. El parámetro INSTANCE_GROUPS especifica el nombre de los grupos utilizando una coma como separador, por ejemplo:

INSTANCE_GROUPS = grupoA, grupoB

Con esta configuración la *instancia* pertenecerá tanto al grupo A como al grupo B.

Si se considera un ambiente *OPS* con cuatro *instancias* agrupadas, y los grupos se configuran como se muestra en la tabla 4.2. *[MAH00]*

Tabla 4.2 Configuración del parámetro INSTANCE_GROUPS del archivo de inicialización.

Instancia	Parámetro INSTANCE_GROUPS
1	grupo12, grupo14, grupo123
2	grupo12, grupo23, grupo123
3	grupo23, grupo123
4	grupo14

Respaldando una base de datos OPS

Respaldar una *base de datos OPS* es muy parecido a respaldar una standalone, sólo hay algunas diferencias que hay que tomar en cuenta. Una *base de datos* puede correr de modo ARCHIVELOG o NOARCHIVELOG. En modo NOARCHIVELOG, los archivos de *redo log* son reciclados y sobrescritos sin ser guardados fuera de línea; por otro lado, de modo ARCHIVELOG, los archivos *redo log* son archivados antes de ser sobrescritos. Si se está utilizando el modo NOARCHIVELOG se corre el riesgo de que se pierda información si es que existe alguna falla. Una recuperación "hasta el último minuto" sólo puede ser realizada si es que se está corriendo en

modo ARCHIVELOG, se tiene que correr en este modo si se quiere contar con alguna de estas opciones:

- Recuperar completamente desde algún punto de falla.
- Ejecutar una recuperación a partir de algún punto en el tiempo.
- Realizar un respaldo en "caliente" o en línea.

Para tener una *base de datos OPS* que corra de modo ARCHIVELOG, se deben de configurar los siguientes tres parámetros en el archivo de inicialización de cada *instancia*:

- LOG_ARCHIVE_START. Habilita el archivado automático. Cuando el parámetro tiene el valor de TRUE, Oracle inicializa el proceso de background ARCH el cual archiva los *redo logs* antes de que sean sobrescritos.
- LOG_ARCHIVE_FORMAT. Especifica el formato que se va a utilizar para nombrar a los archivos. Para una *base de datos OPS* se recomienda utilizar el formato con un número de thread y un número secuencial, así se puede identificar de una manera muy sencilla a que thread pertenecen los archivos. Por ejemplo:

  LOG_ARCHIVE_FORMAT = Prueba_%t_%s.archivelog

  Con esta especificación, Oracle reemplaza %t por el número de thread y %s por un número secuencial, por lo tanto el nombre de los archivos se parecería a los siguientes: Prueba_1_1.archivelog o Prueba_2_23.archivelog.
- LOG_ARCHIVE_DEST. Determina la ruta destino en donde se van a escribir los archivos log. Este parámetro puede tomar uno de los siguientes valores: un filesystem diferente para cada *instancia* el cual sea local para el nodo o un directorio común para todas las *instancias* montado sobre un filesystem el cual sea accesible por todos los nodos. *[STE99]*

Un respaldo en frío en una *base de datos OPS* es muy similar a un respaldo de una *base de datos* standalone, todas las *instancias* que abren la *base de datos* deben ser detenidas; una vez que se han detenido y cerrado la *base de datos*, se puede hacer una copia de los archivos de datos (*datafiles*), de control y de los *redo log*. Si la *base de datos* está corriendo de manera ARCHIVELOG, se puede utilizar la última copia en frío en conjunto con los archivos de *redo log* para poder recuperar la información de todas las transacciones aceptadas por la *base de datos* hasta el punto de falla.

El procedimiento para realizar un respaldo en caliente de una *base de datos OPS* es el mismo que con una *base de datos* standalone. La *base de datos* debe estar corriendo de modo ARCHIVELOG y el respaldo en caliente se realiza cuando la *base de datos* está abierta y corriendo, la *base de datos* va a estar disponible para todas las *instancias* y para todos los usuarios durante el periodo de respaldo. Se deben de seguir los siguientes pasos para realizar este tipo de respaldo:

- Para cada *tablespace* se deben de repetir estos pasos:
  a) Se tiene que cambiar el *tablespace* a modo BACKUP, mediante la siguiente instrucción:

     ALTER TABLESPACE nombre_del_tablespace BEGIN BACKUP;

  b) Respaldar los *datafiles* del *tablespace* utilizando cualquier herramienta del sistema operativo.
  c) Restablecer el *tablespace*, quitarle el modo BACKUP

     ALTER TABLESPACE nombre_del_tablespace END BACKUP;
- Respaldar el archivo de control, con el siguiente comando:

  ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
- Ejecutar la siguiente sentencias en todos los threads para poder respaldar los archivos *redo log*:

  ALTER SYSTEM ARCHIVELOG CURRENT;

Migrando una base de datos a OPS.

Para migrar o convertir una *base de datos* standalone a una OPS se deben de seguir los siguientes pasos:

- Habilitar la opción de OPS, esto se realiza poniendo en el archivo de inicialización el parámetro PARALLEL_SERVER = TRUE para todas las *instancias* que van a abrir la *base de datos.*
- Crear *segmentos de rollback* adicionales para cada *instancia*, y utilizar el parámetro ROLLBACK_SEGMENTS para asociar los *segmentos de rollback* a cada *instancia.*
- Crear hilos o threads de *redo log* para cada *instancia.*
- Apagar la *instancia* que creo la *base de datos.*
- Inicializar el GMS en todos los nodos.
- Inicializar todas las *instancias.*

Por lo menos se debe de crear un *segmento de rollback* para cada *instancia* de la *base de datos.* Se recomienda crear más *segmentos de rollback* para cada *instancia*, dependiendo del número de usuarios y de transacciones que se esperan que se maneje en cada una.

Los archivos de *redo log* están organizados en threads. Cada instancia OPS necesita un thread de *redo log* para poder abrir la *base de datos.* Por lo tanto, se deben de crear threads de *redo log* en línea para las *instancias.*

Para poder agregar mas threads se utiliza el comando *ALTER DATABASE ADD LOGFILE.* Por ejemplo, el siguiente comando crea el thread 4:

ALTER DATABASE ADD LOGFILE THREAD 4

GROUP 10 ('/dev/redo10a_db_ops', '/dev/redo10b_db_ops') SIZE 300M,

GROUP 11 ('/dev/redo11a_db_ops', '/dev/redo11b_db_ops') SIZE 300M,

GROUP 12 ('/dev/redo12a_db_ops', '/dev/redo12b_db_ops') SIZE 300M;

Un thread debe de ser activado antes de que pueda ser utilizado por cualquier *instancia* de la *base de datos*. Un thread puede ser activado público o privado, para activar un thread de forma pública se utiliza el siguiente comando:

ALTER DATABASE ENABLE PUBLIC THREAD 4;

Y para activarlo de forma privada:

ALTER DATABASE ENABLE THREAD 4;

Si un thread es activado público, este podrá ser utilizado por cualquier *instancia*, de lo contrario, si es activado de forma privado, se debe de especificar la instancia que va a utilizar este thread, lo cual se realiza en el archivo de parámetros de inicialización de cada *instancia*. Cuando una *instancia* abre una *base de datos*, adquiere el thread que se encuentre definido por el parámetro *THREAD*, por ejemplo en el archivo initPrueba2.ora se agrega la siguiente línea:

THREAD = 4 # Esta *instancia* utiliza el thread 4

Definición de las pruebas

ARGOS es un sistema de administración de contratos y servicios de comunicaciones analógicos y digitales como puede ser una troncal o un enlace dedicado con fibra óptica. A este sistema ingresan alrededor de 600 usuarios concurrentes en toda la República Mexicana. También existe un área que se encarga especialmente de capturar, validar y emitir dichos contratos; para ellos es muy importante capturar el mayor número de contratos en una jornada, ya que esto les

puede beneficiar económicamente. Por esto es que se requiere que el sistema tarde lo menos posible en realizar las validaciones requeridas para cada contrato. Actualmente se manejan 179 reglas (queries), las cuales detectan los diferentes errores que pueda tener un contrato los cuales evitan que se puedan instalar los servicios y facturarlos.

Los dos equipos en los cuales se van a realizar las pruebas son los siguientes:

- Un equipo SUN con procesador a 360Mhz, 128 MB de memoria RAM, sistema operativo Solaris 8, el manejador de bases de datos es Oracle 8.1.7.
- Un equipo SMP, específicamente es una máquina HP9000/T600 la cual tiene 10 procesadores en paralelo y 2 GB de memoria RAM, el sistema operativo utilizado es HPUX 11.0 y el manejador de bases de datos es Oracle 8.1.7.

PROPUESTA

Se propone que se implemente el paralelismo sobre las reglas de validación que se tienen que ejecutar a cada uno de los contratos para que se realicen en un menor tiempo.

Reporte de resultados obtenidos.

Se realizaron seis pruebas diferentes las cuales fueron seleccionadas por las características de los servicios que se encuentran en los diferentes contratos y la manera en que normalmente se realizan las validaciones sobre los mismos.

En estas pruebas se puede ver la diferencia de tiempos en los que se obtiene el resultado de la aplicación de las reglas de validación sobre ciertos contratos con diferentes servicios. De las cuales los resultados fueron los siguientes:

Prueba 1. Se realizó la validación de un solo contrato

Sesión	Servicios	Tiempo en SMP	Tiempo en Standalone
1	403	01:44.19	2:53.93

Resultado. Esta prueba se realizó para poder comparar la velocidad en que responden los equipos ejecutando la misma prueba; con lo cual vemos que el equipo SMP lo realizó un 40.09% más rápido que el standalone.

Prueba 2. Se realizó la validación de un mismo contrato en cuatro sesiones simultáneas.

Sesión	Servicios	Tiempo en SMP	Tiempo en Standalone
1	403	03:25.82	7:59.56
2	403	03:32.84	8:15.09
3	403	03:28.69	8:17.65
4	403	03:28.41	8:00.00

Resultado. En esta prueba es donde se empieza a ver el resultado de la ejecución en paralelo, ya que se están haciendo 4 peticiones al mismo tiempo, dando como resultado en promedio un 57.18% más rápido en el equipo SMP.

Prueba 3. Se realizó la validación de cuatro contratos en una sesión de manera consecutiva.

Sesión	Servicios	Tiempo en SMP	Tiempo en Standalone
1	47	20.04	57.52
	48	18.90	41.57
	48	18.06	1:03.98
	48	17.48	41.00

Resultado. Aquí se utilizaron contratos que tienen menos servicios que las pruebas anteriores, pero se están ejecutando de manera secuencial, esto para probar como es que se agilizan las consultas utilizando paralelismo sobre los queries, obteniendo los resultados un 62.20% más rápido.

Prueba 4. Se realizó la validación de cuatro contratos diferentes en sesiones simultáneas.

Sesión	Servicios	Tiempo en SMP	Tiempo en Standalone
1	47	41.57	2:48.81
2	48	41.55	1:41.48
3	48	38.08	1:43.16
4	48	37.80	1:42.09

Resultado. Ahora se realizó la misma prueba, pero con la variante de que los contratos se ejecutan en sesiones diferentes simulando algunos usuarios concurrentes, se obtuvo la respuesta un 65.12% más rápido en el equipo SMP.

Prueba 5. Se realizó la validación de siete contratos diferentes en una sesión de manera consecutiva.

Sesión	Servicios	Tiempo en SMP	Tiempo en Standalone
1	11	11.64	1:07.87
	58	22.39	1:04.69
	80	25.88	46.86
	143	43.18	1:42.22
	160	44.47	1:11.04
	300	1:32.14	2:36.92
	403	1:45.54	2:45.66

Resultado. En esta prueba se utilizó una muestra de contratos con diferente número de servicios que se corrieron de manera secuencial, del cual se obtuvo un resultado que se ejecutó un 56.67% más rápido en el equipo SMP.

Prueba 6. Se realizó la validación de siete contratos en sesiones simultáneas.

Sesión	Servicios	Tiempo en SMP	Tiempo en Standalone
1	11	14.15	4:54.50
	58	28.74	5:48.59
	80	37.75	8:16.72
	143	1:07.41	11:03.02
	160	1:10.19	9:24.19
	300	2:16.45	11:16.71
	403	2:56.33	15:25.05
2	11	14.37	4:54.51
	58	31.09	5:48.61
	80	37.79	8:16.89
	143	1:07.96	11:03.06
	160	1:15.41	9:24.24
	300	2:24.28	11:16.76
	403	3:01.99	15:24.24
3	11	14.63	4:54.49
	58	32.82	5:48.61
	80	40.38	8:16.76
	143	1:07.17	11:03.12
	160	1:14.94	9:24.24
	300	2:30.50	11:16.76
	403	3:03.13	15:24.16
4	11	14.14	5:12.76
	58	32.10	5:30.57
	80	38.98	8:16.76
	143	1:09.62	11:03.12
	160	1:14.88	9:24.25
	300	2:23.28	11:16.76
	403	3:01.05	15:24.33

Resultado. El objetivo es probar los equipos con una mayor carga de trabajo, ejecutando la misma prueba 5, pero ahora en cuatro sesiones simultáneas. Las validaciones se ejecutaron un 87.74% más rápido en el equipo SMP.

En todos los casos el resultado fue obtenido en un menor tiempo utilizando el procesamiento en paralelo con un promedio de 61.50% más rápido.

Las corridas de las pruebas completas se encuentran en el disco que acompaña este trabajo, los resultados presentados en las tablas anteriores son un extracto de los mismos para poder facilitar la comparación de los resultados.

Conclusiones

En base a los ejercicios realizados en el sistema de contratación y mercadotecnia (ARGOS) se demuestra que la ejecución en paralelo que se implementó cumplió con las expectativas planteadas, ya que se mejoró el tiempo en que se obtienen los resultados. Las características de funcionamiento que se observaron fueron las siguientes:

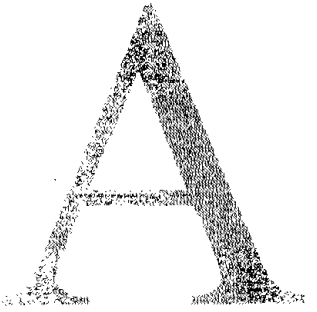
- Mejor tiempo de respuesta. Al utilizar un equipo SMP que tiene 10 procesadores trabajando simultáneamente la base de datos pudo distribuir de una mejor manera el trabajo requerido para lograr terminar las peticiones en un tiempo menor.
- Mayor carga de trabajo. Al aceptar el trabajo de un número mayor de usuarios interactuando con la base de datos al mismo tiempo, gracias a que cada procesador atiende una petición diferente.
- La disponibilidad del equipo. Gracias a que los procesadores realizan tareas independientes entre ellos se pueden procesar más tareas y esto brinda una mayor disponibilidad del equipo para poder atender otras peticiones.

Actualmente algunos de los cambios sobre los queries que se ejecutaron sobre el equipo SMP, fueron aprobados para ser utilizados en el ambiente de producción y esto ha ayudado a los usuarios ya que pueden capturar más contratos y aumentar su productividad en sus áreas de trabajo.

No se debe de perder de vista que al querer utilizar el procesamiento paralelo se realizan procesos adicionales los cuales se encargan de separar la ejecución y posteriormente agrupar los resultados, lo cual no es recomendable cuando se está trabajando con poca información. Sin embargo, el procesamiento en paralelo es la mejor opción para empresas que tienen un gran número de información y que requieren interactuar con ella de forma efectiva, también es

importante realizar un análisis y elaboración de pruebas para poder afinar adecuadamente la base de datos para que brinde los resultados esperados.

Glosario



Definición o explicación de algunos términos utilizados durante el presente trabajo.

B

Base de datos.

Conjunto de datos relacionados que se almacenan de forma que se pueda acceder a ellos de manera sencilla, con la posibilidad de relacionarlos y ordenarlos en base a diferentes criterios.

Binary Large Object (BLOB).

Es un tipo de dato el cual puede almacenar información binaria.

Block.

Un bloque o block es una unidad de almacenamiento en una base de datos Oracle muy pequeña. El bloque de la base de datos se conforma de un encabezado de la información del mismo bloque así como los datos o código PL/SQL. El tamaño del bloque es configurable con un mínimo de 2 Kb y un máximo de 16 Kb.

Buffer.

Este término se refiere a la cantidad de memoria utilizada para almacenar datos. Un buffer almacena datos que van a ser utilizados o que acaban de ser utilizados. En muchos casos, los buffers son copias en memoria de datos que se encuentran en disco. Los buffers pueden ser utilizados como copias de datos para un acceso más rápido, ellos pueden ser modificados y escritos en disco, o pueden ser creados en memoria como almacenamiento temporal. En Oracle, los buffers de la SGA de la base de datos almacenan los blocks recientemente utilizados. El conjunto de buffers de blocks de la base de datos es conocido como el Buffer Cache. Los buffers utilizados para almacenar temporalmete entidades de redo hasta que son escritos en disco es conocido como redo log buffer.

Buffer Cache.

Almacena datos en memoria para que los usuarios tengan una vista de ellos y sus cambios. De este modo, los usuarios nunca hacen cambios directamente en los archivos de la base de datos (datafiles). En cambio Oracle lee los datos y los almacena en memoria para que el proceso del usuario los modifique, y escribe estos cambios en disco algún tiempo después.

Bus.

Conjunto de dispositivos de conexión utilizados por los distintos componentes de una computadora para intercambiar datos e información. Se caracterizan por su capacidad y los elementos que unen, clasificándose en bus de direcciones, bus de datos, bus de entrada/salida, etcétera.

C

Cache.

Un caché es una área de almacenamiento utilizado para obtener un acceso más rápido a los datos. En términos de hardware, el caché es una pequeña cantidad de memoria que es mucho mas rápida que la memoria principal (RAM). Esta memoria es utilizada para reducir el tiempo que se necesita para recargar datos o instrucciones frecuentemente utilizados en el CPU. En Oracle, los buffers de block y la shared pool son considerados cache porque son empleados para almacenar datos e instrucciones para un acceso rápido.

Checkpoint.

Un checkpoint es una operación que forza todo cambio, blocks de datos en memoria que deben ser escritos en disco. Esta es una clave para saber cuanto se toma una base de datos para restablecer en caso de alguna falla.

Clean Buffer.

Un buffer limpio o clean es un buffer que no ha sido modificado. Como este buffer no ha tenido cambios, no es necesario que el proceso DBWR escriba este buffer en disco.

Carácter Large Object (CLOB).

Es un tipo de dato el cual puede almacenar datos de tipo caracter.

Concurrencia.

Este término se refiere a la capacidad de ejecutar varias funciones al mismo tiempo. Oracle provee de concurrencia al permitir que varios usuarios acecen a la base de datos simultáneamente.

Constraint.

Es el mecanismo que asegura que cierta condición relacionada a alguna columna o tabla sea mantenida y verificada.

Control files.

Los Archivos de control contienen la información necesaria para mantener y verificar la integridad de la base de datos

Cuello de botella.

En términos computacionales un cuello de botella es un componente del sistema que limita el rendimiento del mismo sistema.

D

Data Manipulation Language (DML).

Son un conjunto de sentencias las cuales nos permiten manipular los datos de la base de datos.

Data Definition Language (DDL).

Son un conjunto de sentencias las cuales nos permiten definir, mantener y eliminar objetos.

Datafile.

Son los archivos que contienen los datos actuales de la base de datos. Estos datos son tanto los del usuario como los datos del diccionario de datos.

DBMS.

El sistema administrador de la base de datos o database management system es el software y conjunto de herramientas empleadas para administrar la base de datos. El software Oracle es un DBMS.

Deadlock.

Un deadlock ocurre cuando dos o más procesos utilizan un recurso que el otro necesita. Ninguno de los procesos va a dejar de utilizar su recurso hasta que se libere el otro recurso; por eso, ninguno de los procesos puede proceder.

Delete.

Sentencia SQL utilizada para borrar renglones de una tabla.

Diccionario de datos.

Es un conjunto de tablas que utiliza Oracle para mantener información acerca de la base de datos. El diccionario de datos contiene la información acerca de tablas, índices, procedimientos, constrainsts y cualquier objeto de la base de datos.

Dirty Buffer.

Un buffer sucio o dirty es un buffer que ha sido modificado. Es trabajo del proceso DBWR el eventualmente escribir los buffers sucios a disco.

F

Full scan.

Es un método para recorrer la información de forma secuencial.

H

Hint.

Es una sugerencia de como se tienen que realizar las operaciones SQL, en lugar de tomar las opciones predeterminadas.

I

Índice.

Es un recurso diseñado para proveer de un acceso de mayor velocidad a los datos. Un índice evita tener que leer secuencialmente los datos para encontrar lo que se este buscando.

Índice particionado.

Es un índice que se divide a partir de ciertas consideraciones para facilitar la búsqueda sobre el mismo.

Insert.

Sentencia SQL utilizada para insertar renglones de una tabla.

Instancia.

Es una combinación entre procesos de background y estructuras de memoria. La instancia debe de estar inicializada para poder acceder a la base de datos. Cada vez que una instancia es inicializada es alojada una área de sistema global de memoria (SGA) y procesos de background.

J

Join.

Un query que selecciona datos de mas de una tabla. Los datos seleccionados de diferentes tablas son determinados por las condiciones especificadas en la cláusula WHERE de la sentencia.

L

LOB.

Es un tipo de dato objeto (large object datatype) que puede almacenar hasta 4Gb de información. Este tipo de dato se clasifica en tres: CLOB, NCLOB Y BLOB

Log Buffer.

Almacena información especial llamada redo, la cual ayuda a Oracle para reconstruir los cambios de datos hasta el momento de falla del sistema.

M

Massively Parallel Processor (MPP).

Una computadora multiprocesador consiste en varios procesadores independientes que se comunican a través de un bus de alta velocidad.

N

NCLOB.

Es un tipo de dato el cual puede almacenar caracteres de lenguaje nacional.

O

Oracle Parallel Server (OPS).

Es una herramienta la cual nos permite utilizar varias computadoras con un subsistema de discos compartidos para acceder a la misma base de datos.

ORDBMS

Es un sistema manejador de base de datos relacionada a objetos, del inglés "Object Relational DataBase Management System". Este sistema almacena objetos utilizados por las aplicaciones en una base de datos relacional. Cuando las aplicaciones utilizan este tipo de bases de datos, normalmente manejan datos los cuales son almacenados en forma de objetos. Sin embargo, el sistema transforma los datos del objeto y los introduce a tablas con renglones y columnas para manejarlos como una base de datos relacional.

Q

Query.

Un query es una transaccion de solo lectura ejecutada en la base de datos. Un query es generado utilizando la sentencia SELECT. Generalmente los usuarios distinguen entre queries y otras transacciones porque un query no afecta los datos en una base de datos.

R

RDBMS.

El sistema administrador de la base de datos relacionales o relational database management system es un DBMS que es relacional. Es decir, que la forma de accesar a los datos se realiza de una manera relacionada. Oracle es un RDBMS.

Redo log file.

El archivo que contiene una copia de todos los blocks de datos que han sido modificados como resultado de una transacción de la base de datos. Si existe cualquier falla, una transacción puede ser restablecida por estos redo blocks.

Rollback.

Es el acto de deshacer cambios que se habían realizado por una transacción.

Rollback segment.

Es el lugar en la base de datos donde se guarda la información para deshacer y puede ser obtenida si se ejecuta un rollback.

S

Shared Pool.

Almacena muchos artículos que son de "misión crítica" para la operación de la base de datos Oracle. Los componentes de la shared pool incluyen la librería caché, para almacenar las instrucciones SQL parseadas para poder ser reutilizadas por los usuarios; el diccionario o caché de renglón, para almacenar en memoria alguna información del diccionario de datos para agilizar su acceso.

Sincronizar.

Hacer que coincidan en el tiempo dos o mas movimientos o fenómenos.

System Global Area (SGA).

Es un grupo de estructuras de memoria compartidas que contienen datos e información de control para una instancia de una base de datos Oracle.

T

Tarea.

Es un conjunto de procesos para realizar un objetivo.

Tabla.

Es la unidad básica de almacenamiento en una base de datos. Los datos son almacenados en tablas.

Tablespace.

Es una estructura lógica que consiste en uno o más datafiles. Un tablespace es empleado lógicamente para alojar objetos como tablas e índices.

Transacción.

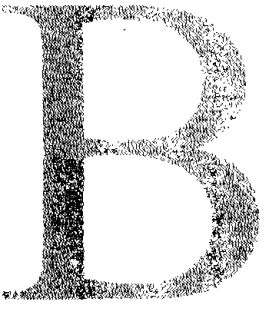
Es una unidad lógica de trabajo que consta de una o muchas instrucciones SQL ejecutadas por un usuario. Conforme al estándar SQL ANSI/ISO una transacción comienza con la primera sentencia SQL ejecutada por un usuario y termina cuando ese mismo usuario ejecuta la instrucción COMMIT o ROLLBACK.

U

Update.

Sentencia SQL utilizada para actualizar el valor de las columnas en una tabla.

Pruebas



En este apéndice se encuentra el código fuente de las funciones utilizadas para realizar las pruebas; así como un extracto de los archivos de salida obtenidos durante las mismas.

En ellos se pueden observar los resultados de los queries utilizados y los respectivos tiempos de ejecución de cada proceso.

CD-ROM Anexo

En el CD-ROM anexo a este trabajo se encuentran los archivos fuente utilizados; así como los archivos de salida generados por cada una de las pruebas realizadas. El árbol de directorios es el siguiente:

/Funciones

/Resultados

/Paralelo

/Prueba 1

/Prueba 2

/Prueba 3

/Prueba 4

/Prueba 5

/Prueba 6

/Serie

/Prueba 1

/Prueba 2

/Prueba 3

/Prueba 4

/Prueba 5

/Prueba 6

Los diferentes tipos de archivo son:

- Pl. Funciones utilizadas para la realización de las pruebas.
- Sql. Guiones utilizados para la ejecución de las pruebas.
- Lst. Resultados obtenidos de la ejecución del guión.

Ejecución de las Pruebas

Antes de iniciar las pruebas se preparó el ambiente para poder obtener los datos suficientes al momento de ejecución de las mismas, como es el tiempo en que se realizó y guardar los resultados obtenidos.

Para poder ejecutar las pruebas se crearon scripts en los cuales se invoca el procedimiento "REGLASVAL" ya sea de forma secuencial o en paralelo en diferentes sesiones de la base de datos.

El procedimiento de ejecución de las pruebas fue el siguiente:

1. Abrir una o varias sesiones de la base de datos según sea el caso.
2. Preparar el ambiente en cada una de las sesiones.
   - Set timing on
   - Spool Resultado.lst
3. Ejecutar el script.
   - @script.sql
4. Cerrar el archivo de salida.
   - Spool off

Código Fuente

El primer procedimiento utilizado en el presente trabajo se realizó para poder ejecutar las reglas de validación utilizadas para controlar la consistencia de los datos que se encuentran en un contrato antes de que se pueda empezar a facturarlo. Este procedimiento es llamado "REGLASVAL" y su código es el siguiente:

```
CREATE OR REPLACE PROCEDURE REGLASVAL
(lFOLIOCTR IN number) is

 FolioValCtr number;
 Cont        number          default 0;
 ContErr     number          default 0;
 sQueryRegla varchar2(3000);
 sRespRegla1 varchar2(500);
 sRespRegla2 varchar2(500);
 sRespRegla3 varchar2(500);
 sRespRegla4 varchar2(500);
 sRespRegla5 varchar2(500);

 type tRegla is record (
```

```
      Cod_Regla  Regla_Emision_Contrato.Cod_Regla_Emision_Ctr%type,
      Tipo_Regla Regla_Emision_Contrato.Tipo_Regla_Emision_Ctr%type,
      Mensaje    Message.Lib_msg%type,
      Query      Regla_Emision_Contrato.Query_Sql_Regla_Emision_Ctr%type,
      Num_Campos Regla_Emision_Contrato.Num_Campos_Query_Regla_Ctr%type);
 Regla tRegla;

 TYPE ReglaCurTyp IS REF CURSOR;
 cQueryRegla ReglaCurTyp;

 cursor cReglas is
       SELECT REC.Cod_Regla_Emision_Ctr,
              REC.Tipo_Regla_Emision_Ctr,
              MES.Lib_msg,
              REC.Query_Sql_Regla_Emision_Ctr,
              REC.Num_Campos_Query_Regla_Ctr
         FROM Message MES,
              Regla_Emision_Contrato REC
        WHERE MES.Code_Msg = REC.Cod_Msj
        ORDER BY REC.Cod_Regla_Emision_Ctr;

BEGIN

SELECT Folio_Validacion_Ctr
  INTO FolioValCtr
  FROM Validacion_Contrato VC
 WHERE Folio_ctr = lFOLIOCTR
   AND Fecha_creacion_validacion_ctr =
           (SELECT MAX(Fecha_creacion_validacion_ctr)
              FROM Validacion_Contrato
             WHERE Folio_ctr = lFOLIOCTR);

dbms_output.put_line('Folio_Validacion_Ctr: ' || FolioValCtr);

if FolioValCtr = NULL then
   dbms_output.put_line('Obtiene Folio Nuevo');
   SELECT VCTR_SEQ.NEXTVAL
     INTO FolioValCtr
     FROM Dual;

   dbms_output.put_line('Folio_Validacion_Ctr Nuevo: ' || FolioValCtr);

   INSERT INTO Validacion_Contrato
         (Folio_Validacion_ctr,
          Folio_Ctr,
          Fecha_Validacion_Ctr,
          Fecha_Rechazo_Ctr,
          Es_Emitido_Ctr)
   VALUES(FolioValCtr,
          lFOLIOCTR,
          SYSDATE,
          NULL,
          'N');
end if;


dbms_output.put_line('Borra Error_Validacion_Contrato');

DELETE Error_Validacion_Contrato
 WHERE Folio_Validacion_Ctr = FolioValCtr;

if sql%notfound then
   dbms_output.put_line('No hay Renglones');
end if;

dbms_output.put_line('Abre Cursor de Reglas');
OPEN cReglas;

loop

   FETCH cReglas
```

```
    INTO Regla;
    EXIT WHEN cReglas%notfound;

    Cont := Cont+1;

    dbms_output.put_line('Regla ' || Cont || ' Cod Regla [' || Regla.Cod_Regla || ']');

    sQueryRegla := RemplazaCadena(Regla.Query, lFOLIOCTR);

    Begin
       dbms_output.put_line('Ejecuta Regla');
       OPEN cQueryRegla FOR sQueryRegla;
       loop

          dbms_output.put_line('Obtiene datos de la Regla');
          if Regla.Num_Campos = 1 then
             FETCH cQueryRegla
              INTO sRespRegla1;
          elsif Regla.Num_Campos = 2 then
             FETCH cQueryRegla
              INTO sRespRegla1,
                   sRespRegla2;
          elsif Regla.Num_Campos = 3 then
             FETCH cQueryRegla
              INTO sRespRegla1,
                   sRespRegla2,
                   sRespRegla3;
          elsif Regla.Num_Campos = 4 then
             FETCH cQueryRegla
              INTO sRespRegla1,
                   sRespRegla2,
                   sRespRegla3,
                   sRespRegla4;
          elsif Regla.Num_Campos = 5 then
             FETCH cQueryRegla
              INTO sRespRegla1,
                   sRespRegla2,
                   sRespRegla3,
                   sRespRegla4,
                   sRespRegla5;
          end if;
          EXIT WHEN cQueryRegla%notfound;

       End loop;
       if sRespRegla1 is not Null then
          ContErr := ContErr + 1;
       end if;
    Exception
       When no_data_found then null;
    End;
end loop;

dbms_output.put_line('Cierra el Cursor cReglas');
close cReglas;
dbms_output.put_line('Num Errores: ' || ContErr);

END;
/
```

Este procedimiento a su vez hace mención a una función llamada "RemplazaCadena" la cual se utiliza para reemplazar el símbolo % por el valor de la variable que se está evaluando en ese momento, su código es el siguiente:

```
CREATE OR REPLACE FUNCTION REMPLAZACADENA
(sQueryFuente IN varchar2, FolioCtr IN number)
return varchar2 is

 Cont        number         default 0;
 IniCad      number         default 0;
 Encontrado number(1)       default 0;
 sQueryRet varchar2(3000) default Null;

BEGIN
for Cont in 0..length(sQueryFuente) loop
if substr(sQueryFuente,Cont,1) = '%' and Encontrado = 0 then
Encontrado := 1;
       IniCad := Cont;
       sQueryRet := substr(sQueryFuente,1,Cont-1) || FolioCtr;
elsif substr(sQueryFuente,Cont,1) = '%' and Encontrado = 1 then
sQueryRet := sQueryRet || substr(sQueryFuente,IniCad+1, Cont-IniCad-1) || FolioCtr;
       IniCad := Cont;
end if;
end loop;
sQueryRet := sQueryRet || substr(sQueryFuente,IniCad+1);
return sQueryRet;
END;
/
```

Bibliografía

[BAM99] BAUER Mark. Oracle8i Parallel Server Concepts & Administration. Ed Oracle Corporation. Estados Unidos. 1999. 454p. Part No. A67778-01

[BAU99] BAUER Mark. Oracle8i Parallel Server Concepts. Ed Oracle Corporation. Estados Unidos. 1999. 216p. Part No. A76968-01

[BAUE9] BAUER Mark. Oracle8i Parallel Server Administration, Deployment, and Performance. Ed Oracle Corporation. Estados Unidos. 1999. 296p. Part No. A76970-01

[BEC96] BECKER Rachel. Oracle Unleashed. Ed Sams. Estados Unidos. 1996. 1404p. ASIN: 067230872X

[COU01] COUCHMAN Jason S. Oracle 8i Certified Professional SQL & PL/SQL Exam Guide. Ed McGraw Hill. Estados Unidos. 2001. 570p. ISBN: 0-07-2191-53-8.

[ERN01] ERNST, Bruce, Ramussen Hanne Rue, Schwinn Ulrike, Venkatachalam Vijay. Architecture and Administration. Ed. Oracle Corporation. Estados Unidos. 2001. 878p. Part No: 30049GC11.

[GRE01] GREENWALD, Rick, Stackowiak Robert, Stern Jonathan. Oracle Essentials. Ed. O'Reilly. Estados Unidos. 2001. 364p. ISBN: 0-596-00179-7.

[LEV99] LEVERENZ Lefty, Rehfield Diana, Baird Cathy. Oracle8i Concepts. Ed Oracle Corporation. Estados Unidos. 1999. 902p. Part No. A76965-01

[MAH00] MAHAPATRA Tushar. Oracle Parallel Processing. Ed O'Reilly. Estados Unidos. 2000. 268p. ISBN: 1-56592-701-X

[RIC99] RICH Kathy. Oracle8i Utilities. Ed Oracle Corporation. Estados Unidos. 1999. 432p. Part No. A76955-01

[STE99] STAINER, Deborah. Oracle8i Parallel Server Setup and Configuration Guide. Ed Oracle Corporation. Estados Unidos. 1999. 278p. Part No. A76934-01

[URB00] URBANO Randy. Oracle8i Migration. Ed Oracle Corporation. Estados Unidos. 2000. 496p. Part No. A86632-01

[WHA97] WHALEN, Edward, Deluca Steve Adrien. Teach Yourself Oracle 8 in 21 Days. Ed Sams Publishing. Estados Unidos 1997. 604p. ISBN: 0672311593