Servicio Andaluz de Salud
CONSEJERÍA DE SALUD

Oficina Técnica para la Gestión y Supervisión de Servicios TIC
Subdirección de Tecnologías de la Información

Best practices para Dataware Housing sobre Real Application Cluster

Referencia documento: InfV5_JASAS_DWH_vs_RAC_v920.doc
Fecha: 16/11/2018
Versión: 9.2.0

Registro de Cambios

Fecha	Autor	Versión	Notas
14/07/2011	Oracle ACS	2.3.0	Versión inicial
13/10/2011	Oracle ACS	2.4.0	Revisión Octubre de 2011
14/03/2013	Oracle ACS	3.1.0	Revisión Enero de 2012
14 de Marzo de 2013	Oracle ACS	4.1	Revisión de Marzo de 2013, contrato 2012-2014
13 de Junio de 2013	Oracle ACS	4.2	Revisión de Junio de 2013, contrato 2012-2014
17 de Octubre de 2013	Oracle ACS	4.3	Revisión de Octubre de 2013, contrato 2012-2014
16 de Julio de 2015	Enrique Ramiro	6.1	Revisión de Julio de 2015, contrato 2014-2016
16 de Diciembre de 2015	Enrique Ramiro	6.2	Revisión de Diciembre de 2015, contrato 2014-2016
16 de Junio de 2016	Enrique Ramiro	7.1	Revisión de Junio de 2016, contrato 2014-2016
16 de Noviembre de 2016	Enrique Ramiro	7.2	Revisión de Noviembre de 2016, contrato 2014-2016
16 de Junio de 2017	Enrique Ramiro	8.1	Revisión de Junio de 2017, contrato 2016-2018
16 de Noviembre de 2.017	Enrique Ramiro	8.2	Revisión de Noviembre de 2017, contrato 2016-2018
16 de Junio de 2.018	Enrique Ramiro	9.1	Revisión de Junio de 2018, contrato 2016-2018
16 de Noviembre de 2.018	Enrique Ramiro	9.2	Revisión de Noviembre de 2018, contrato 2016-2018

Revisiones

Nombre	Role
Jonathan Ortiz	Advanced Support Engineer
Gregorio Adame	Advanced Support Engineer
José María Gómez	Technical Account Manager

Distribución

Copia	Nombre	Empresa
1	Subdirección de Tecnologías de la Información	Servicio Andaluz de Salud, Junta de Andalucía
2	Dirección General de Política Digital	Consejería de Hacienda y Administración Pública, Junta de Andalucía

Índice de Contenidos

Control de cambios

Cambio	Descripción	Página
1	No se realizan cambios en esta versión	N/A

Introducción

Este documento recoge una serie de recomendaciones de Oracle Soporte planteadas como buenas prácticas de desarrollo para entornos de Dataware Housing que hagan uso de Oracle RDBMS 12cR2 Real Application Cluster (RAC).

Estas recomendaciones están encaminadas a minimizar los posibles problemas de rendimiento en sistema de cualquier tamaño y en la gran mayoría de los casos se basan en la experiencia de casos reales gestionados por Oracle Soporte.

Finalmente, este documento también recoge una serie de conceptos de componentes, módulos y tecnologías relacionadas con Oracle RDBMS 12cR2 Real Application Cluster (RAC) y con bases de datos de tipo Datawarehouse que, a juicio de Oracle Soporte, deberían tenerse claros para asegurar la aplicación de las recomendaciones recogidas en este documento y, de manera general, entender los productos Oracle sobre los que se sustentan los sistemas y aplicaciones.

Objetivos de este documento

A lo largo de los puntos de este documento se irá definiendo una guía de buenas prácticas para entornos de Dataware Housing sobre bases de datos en clúster a través de Oracle RDBMS Real Application Cluster (RAC).

Esta guía contendrá tanto prácticas recomendadas como prácticas a evitar y se apoyará en ejemplos y en información que permita analizar las recomendaciones en cada uno de los entornos de desarrollo y preproducción.

Este documento se centra principalmente en las versiones Oracle RDBMS Real Application Cluster 12cR2 y 12cR1, aunque la mayoría de las recomendaciones son igualmente aplicables a las versiones 11g y algunas de ellas a la 10g y 9iR2.

Best Practices para Data Warehousing sobre Oracle RAC

Introducción

Hoy en día, el uso de Data Warehousing es muy común en bases de datos Oracle pero se sigue evolucionando para dar respuesta al incremento de las elevadas demandas de negocio y la mejora de las capacidades técnicas tanto de software como de hardware. En paralelo a esta evolución, Oracle RAC ha seguido creciendo y se ha consolidado como una de las arquitectura más viables, de alta disponibilidad, escalable que puede soportar diversas necesidades.

La unión de ambos, Data Warehousing y Oracle RAC, ha llegado a ser muy popular dentro de los sistemas Data Warehousing y su demanda se ha incrementado elevadamente. Estos requisitos han encontrado un hueco en la flexibilidad y la potencia de Oracle RAC.

Warehousing, hoy

Data Warehousing ha evolucionado desde aquellos días en los que unas pocas actualizaciones mensuales de esquemas en estrella satisfacían una o dos necesidades de negocio estrictamente predefinidas. En la actualidad se ofrece acceso a datos e información en gran variedad de estructuras, dentro y fuera de la base de datos, a través de numerosas vías de acceso, con actualizaciones que tienen planificaciones regulares e irregulares. Finalmente, no se está lejos de ofrecer disponibilidad 24*7 y acceso en tiempo real a las demandas de negocio cada vez mayores realizadas por los usuarios y la comunidad empresarial en general.

Los sistemas Data Warehouse de Oracle RAC están demostrando ser una estrategia común para satisfacer estas demandas basándose en las tres principales características:

- Oracle Partitioning
- Oracle Real Application Clusters (RAC), y
- Operaciones paralelas

Estas tres características son la clave del éxito del Data Warehouse moderno. La implementación de estas características utilizando buenas prácticas para soportar las demandas de los sistemas de Data Warehouse es de gran importancia. Por lo tanto, es esencial comprender los aspectos relevantes de estas características en el contexto de Data Warehouse en un entorno Oracle.

Best practices

Las consideraciones clave en el diseño de Data Warehouse con Oracle RAC incluyen

- Oracle Partitioning
- Ejecución en paralelo
- Gestión de la carga de trabajo.

La relevancia de cada una de estas características depende de la implementación específica. En esta sección, se presentan una serie de buenas prácticas centradas en los aspectos técnicos de las características de Data Warehouse que son relevantes para Oracle RAC.

Después de esto, se continuará con una discusión acerca de las consideraciones que deberían tenerse en cuenta a la hora de la implementación.

Oracle Partitioning

Fundamentos de Oracle Partitioning

Oracle partitioning permite dividir una tabla, índice o index-organized table (IOT) en componentes más pequeños. Cada pieza de un objeto de base de datos se llama partición. Cada partición tiene su propio nombre y, opcionalmente, puede tener sus propias características de almacenamiento, tales como tener habilitada la compresión de tablas o almacenarse en diferentes tablespaces.

Desde la perspectiva de un administrador de bases de datos, un objeto particionado tiene múltiples componentes que pueden ser gestionados de forma colectiva o individual. Esto le da bastante flexibilidad al administrador en la gestión de objetos particionados.

Sin embargo, desde la perspectiva de la aplicación, una tabla particionada es idéntica a una tabla no particionada; cuando se accede a una tabla particionada usando comandos SQL, no son necesarias modificaciones.

Las tablas particionadas utilizan una clave de particionamiento o *key partitioning*, un conjunto de columnas, que determinan en qué partición reside una fila dada. Oracle proporciona las siguientes estrategias de partitioning:

- Particionamiento simple (de un solo nivel) ó *Single-Level Partitioning*
  - Partitioning por Rango
  - Partitioning por Hash
  - Partitioning por Lista
- Particionamiento compuesto o *Composite Partitioning*
  - Partitioning compuesto Rango-Rango
  - Partitioning compuesto Rango-Hash
  - Partitioning compuesto Rango-Lista
  - Partitioning compuesto Lista-Rango
  - Partitioning compuesto Lista-Hash
  - Partitioning compuesto Lista-Lista
  - Partitioning compuesto Hash-Hash
  - Partitioning compuesto Hash-Lista
  - Partitioning compuesto Hash-Rango
- Extensiones de particionamiento o *Partitioning Extensions*
  - Partitioning por intervalo (a partir de Oracle RDBMS 11g)
    - Simple
    - Compuestos: Intervalo-Rango, Intervalo-Hash, Intervalo-Lista
  - REF Partitioning (a partir de Oracle RDBMS 11g)
  - Partitioning basado en columnas virtuales (a partir de Oracle RDBMS 11g)

- Partitioning por Rango para Hash Clusters (a partir de 12.1.0.2)
- Partitioning Interval-Reference (a partir de 12.1.0.1)
- Auto-List Partitioning (a partir de 12cR2)
- Multi-column List Partitioning (a partir de 12cR2)
- Partitioning de tablas externas (a partir de 12cR2)

Ventajas de Oracle Partitioning

Administración y uso

Con el uso de partitioning, las operaciones de mantenimiento se pueden centrar en determinadas partes de las tablas. Para las operaciones de mantenimiento en un objeto de la base de datos, es posible realizar estas operaciones en función de cada partición, dividiendo así el proceso de mantenimiento en pedazos más manejables. Un uso típico de partitioning para administración es soportar una ventana de desplazamiento o 'rolling window' en un proceso de carga en Data Warehouse.

Disponibilidad

Particionar objetos de la base de datos proporciona independencia en las particiones. Esta característica de independencia en cada partición puede ser una parte importante en la estrategia para la alta disponibilidad. Por ejemplo, si una partición de una tabla particionada no está disponible, las demás particiones de la tabla permanecerán disponibles; la aplicación puede continuar ejecutando consultas y transacciones contra la tabla particionada, y las operaciones de base de datos se ejecutarán correctamente siempre que no necesiten acceder a la partición que no está disponible. El administrador de la base de datos puede especificar que cada partición se almacene en una tablespace separada. Esto le permitirá al administrador hacer operaciones de backup y recovery en cada partición individualmente, independientemente de las otras particiones de la tabla. Por otra parte, el partitioning puede reducir el tiempo necesario en paradas planificadas.

Rendimiento

Al limitar la cantidad de datos examinados o con los que se va a operar, y habilitando la ejecución en paralelo, Oracle Partitioning ofrece una serie de mejoras en el rendimiento. Estas mejoras incluyen:

Partitioning Pruning

Partitioning Pruning es la mejora más sencilla y también el medio más sólido para mejorar el rendimiento con el uso de Partitioning. Dada una consulta sobre una tabla particionada, el optimizador tendrá acceso al mínimo conjunto de particiones al que se deben acceder para resolver la consulta. Partitioning pruning suele mejorar el rendimiento de una consulta en varios órdenes de magnitud. Partitioning pruning es compatible con las demás prestaciones de Oracle y por tanto se usará junto con cualquier otra técnica de indexación, operación de join, o método de acceso en paralelo.

Partition-wise Joins

El Partitioning también puede mejorar el rendimiento de joins en multi-tablas, usando una técnica conocida como Partition-wise join. La técnica de Partition-wise joins puede ser total o parcial. Oracle decide qué tipo de join usar:

Full Partition-wise Join

Un full partition-wise join divide un join grande en joins más pequeños entre pares de particiones procedentes de las dos tablas combinadas. Para utilizar esta función, los dos objetos (tabla, índice o partición) deben ser igualmente particionados según sus claves de join. Ofrece un beneficio significativo en el rendimiento de ejecuciones tanto paralelas como serializadas.

Partial Partition-wise Join

Oracle Database sólo puede realizar partial partition-wise join en paralelo. A diferencia de full partition-wise join, partial partition-wise join necesita una sola tabla en la clave join, no en ambas tablas. La tabla particionada se conoce como tabla de referencia. La otra tabla puede o no estar particionada y por tanto es más común que la full partition-wise join.

Para ejecutar un partial partition-wise join, Oracle particiona dinámicamente la otra tabla en base al partitioning hecho a la tabla de referencia. Una vez que la otra tabla ha sido particionada, la ejecución es similar a la full partition-wise join. La mejora en rendimiento que partial partition-wise join obtiene en joins en tablas no particionadas es que la tabla de referencia no se mueve durante la operación de join.

Cargas con Partition-Exchange

Carga de datos nuevos a través de operaciones de metadata

Otras ventajas encaminadas a mejorar los tres puntos anteriores (Administración y uso, Disponibilidad y Rendimiento):

- *Mantenimiento de partición en particiones múltiples (desde 12cR1)*
- *Mantenimiento de índice global asíncrono (desde 12cR1)*
- *Zone maps (desde 12cR1)*
- *Online partition MOVE (desde 12cR1)*
- *Cascading TRUNCATE (desde 12cR1)*
- *Partial indexing (desde 12cR1)*
- *Operaciones de mantenimiento Online partition (desde 12cR2)*
- *Conversión Online de tabla a tabla particionada (desde 12cR2)*
- *Reducción de invalidaciones de cursor para DDLs (desde 12cR2)*
- *Filtered partition maintenance (desde 12cR2)*
- *Read only partitions (desde 12cR2)*
- *Create table for exchange (desde 12cR2)*

Entrar en detalles de todas las estrategias de partitioning va más allá del alcance de este documento. Sin embargo, tenga en cuenta que serán la gestión y el rendimiento los que impulsarán la decisión de qué estrategia llevar a cabo. Por ejemplo, si el objetivo del partitioning es facilitar la carga y el mantenimiento, una estrategia típica será partitioning por rango de tiempo con índices locales, de esta forma se puede implementar el intercambio y el borrado de particiones. Si el objetivo es el rendimiento, la estrategia de partición se decidirá en función de las consultas más críticas, y se tendrán en cuenta las partition-wise joins (totales y parciales) y el uso de índices globales, como se verá en secciones siguientes.

Arquitectura de Oracle Parallel Execution

La arquitectura de Oracle Parallel Execution (PX) está compuesta por un proceso coordinador de consultas (QC) y un conjunto de procesos PX (también denominados PQ esclavos) que soportan el paralelismo dentro de la operación. Oracle soporta la ejecución en paralelo de todas las operaciones relacionales (por ejemplo, scans, joins, order-by, agregaciones, operaciones en conjuntos, etc.), DML (insert, update, delete), DDL (por ejemplo, create table, index, vistas, vistas materializadas), reorganización de datos (por ejemplo, operaciones de mantenimiento de particiones como move, split, coalesce), carga y descarga a través de tablas externas y extensiones SQL para Analytics y Data-Mining.

Cómo funciona la Ejecución en Paralelo

La ejecución en paralelo divide la tarea de ejecutar una sentencia SQL en varias unidades más pequeñas, donde cada unidad es ejecutada por un proceso independiente. El proceso de usuario que quiere ejecutar una consulta en paralelo toma el rol de coordinador de ejecución paralela o coordinador de consulta (QC). El coordinador de consulta hace lo siguiente:

- Analiza la consulta y determina el grado de paralelismo
- Asigna uno o dos conjuntos de procesos de ejecución en paralelo – hilos o procesos PX (o PQ esclavos)
- Controla la consulta y envía instrucciones a los esclavos PQ.
- Determina que tablas o índices deben ser analizados por los esclavos PQ.
- Proporciona el resultado final para el usuario.

Las sentencias paralelas se ejecutan en los procesos esclavos y pueden ejecutarse en un sólo nodo (paralelismo intra-node) o en múltiples nodos (paralelismo inter-node) a través de la red. La comunicación intra-node utiliza memoria compartida y la comunicación inter-nodo usa el protocolo IPC a través de interconnects de alta velocidad. Una vez que la sentencia ha sido procesada por completo, los procesos de ejecución paralela regresan al pool.

Determinación Dinámica de Unidades de Trabajo

La unidad básica de trabajo en paralelismo se denomina por el término en inglés, granule. Oracle divide los objetos utilizados en operaciones paralelas (recorrido de tablas, actualización de tablas, creación de índices,...) en granules.

Oracle puede utilizar granules por rango de bloques (rango de bloques físicos de una tabla o índice) o gránulos particionados. Los procesos de ejecución paralela ejecutan la operación de un gránulo a la vez. El número de granules y su tamaño se correlacionan con el grado de paralelismo (DOP) y el tamaño del objeto. No hay forma de forzar una estrategia de gránulos específica como Oracle hace internamente. Dependiendo de la operación en paralelo, Oracle o elige el gránulo basado en rango de bloques o una partición completa como estrategia óptima de gránulo.

Operaciones Paralelas y Grado de Paralelismo (DOP)

Una vez que el optimizador determina el plan de ejecución de una sentencia, el coordinador de la ejecución paralela determina el método de paralelización para cada operación del plan de ejecución. El coordinador tiene que decidir si una operación se puede realizar en paralelo y, en caso afirmativo, cuántos procesos de ejecución paralela añadir a la lista. El número de procesos de ejecución paralela utilizados para una operación es el grado de paralelismo (DOP) y se determina en tiempo de ejecución de la consulta, en base a volumen de trabajo actual y otros criterios del sistema, tales como la prioridad de la consulta realizada por un usuario.

El siguiente diagrama ilustra la forma en que funcionan las operaciones paralelas y el DOP:

```
SELECT cust_last_name,cust_first_name
FROM
customers ORDER BY cust_last_name;
```

El plan de ejecución implementa un full scan de la tabla CUSTOMERS seguida de una ordenación de las filas seleccionadas en función de la columna CUST_LAST_NAME. Para facilitar este ejemplo, supongamos que esta columna no está indexada. Suponga también que el grado de paralelismo de la consulta se establece en 3, lo que significa que se pueden activar tres procesos de ejecución paralela para una determinada operación. A cada una de las dos operaciones (recorrido y ordenación) que se realizan concurrentemente se le da su propio conjunto de procesos de ejecución paralela. Por lo tanto, ambas operaciones tienen paralelismo.

La paralelización de una operación individual donde los procesos de ejecución paralela realizan la misma operación sobre conjuntos de filas más pequeños se denomina paralelismo intra-operation. Cuando dos operaciones se ejecutan concurrentemente en diferentes conjuntos de procesos de ejecución paralela con flujo de datos de una operación a la otra, se consigue lo que se denomina paralelismo inter-operation. Debido a la naturaleza productor/consumidor de las operaciones de procesos de Oracle, basta con realizar simultáneamente solo dos operaciones en un árbol determinado para minimizar el tiempo de ejecución.

Cómo se comunican los Procesos de Ejecución Paralela

Generalmente, para ejecutar una consulta en paralelo, Oracle crea un conjunto de procesos productores y un conjunto de procesos consumidores. Los procesos productores recuperan las filas de las tablas y los procesos consumidores llevan a cabo operaciones sobre esas filas, tales como join, ordenación, DML y DDL.

Cada proceso del conjunto de productores tiene una conexión con cada proceso del conjunto consumidor. Esto significa que el número de conexiones virtuales entre procesos de ejecución paralela aumenta al cuadrado del DOP.

Cada canal de comunicaciones tiene al menos una, y en algunos casos hasta cuatro buffers de memoria. Al tener varios buffers de memoria se facilita la comunicación asíncrona entre los procesos de ejecución paralela.

Una sola instancia de entorno utiliza hasta tres buffers para cada canal de comunicación mientras que un entorno Oracle Real Application Cluster utiliza hasta cuatro buffers para cada canal de comunicación. Cuando hay una conexión entre dos procesos en la misma instancia, los procesos se comunican mediante buffers bidireccionales.

Cuando la conexión es entre dos procesos en la misma instancia, los procesos se comunican pasando buffers en memoria (en la shared pool). Cuando la conexión es entre procesos de diferentes instancias, los mensajes se envían utilizando protocolos externos de red de alta velocidad o interconnect. Cada proceso ejecutado en paralelo tiene una conexión adicional al coordinador de ejecución paralela. Es importante dimensionar la shared pool de manera adecuada cuando se usa ejecución paralela. Si no hay suficiente espacio libre en la shared pool para asignar los buffers de memoria necesarios para un PX, no se podrá iniciar.

Combinaciones de Consultas Paralelas y Redistribución

El paralelismo que pueden tener las operaciones se halla en el optimizador; cuando se determina el plan de ejecución de una sentencia, se decide el DOP y el método de paralelización más óptimo para cada operación. Por ejemplo, el método de paralelización podría ser un full table scan por rango de bloque o paralelizar un index range scan por partición. Por otra parte, se determinan los requisitos de redistribución de cada operación. Un requisito de redistribución de una operación es la forma en la que las filas implicadas en la operación se deben dividir o redistribuir entre los procesos de ejecución paralela.

Por defecto, los joins paralelos entre dos tablas sin particionar necesitan que ambas tablas de entrada se redistribuyan según la clave de join y en subconjuntos disjuntos de filas. A estos subconjuntos disjuntos de filas se les hace join de dos en dos por un sólo proceso de ejecución paralela. Esta redistribución de la operación sigue intercambiando filas entre distintos procesos de ejecución paralela. Los posibles métodos de distribución que envían los datos (o redistribuyen) desde procesos productores de consulta a procesos consumidores de consulta son:

- PARTITION: Mapea las filas a procesos de consulta basándose en el partitioning de una tabla o índice.
- HASH: Mapea las filas a procesos de consulta utilizando una función HASH en la clave de join.
- HYBRID HASH: La redistribución hash híbrida, introducida en Oracle Database 12c, es una técnica de distribución adaptativa (*adaptive*) que retrasa la decisión del método de distribución final hasta el tiempo de ejecución y se basa en el tamaño del conjunto de resultados. Comprueba el recuento contra un umbral máximo: (1) si se supera el umbral, es más efectivo en cuanto a coste distribuir las filas según una distribución hash, (2) si el tamaño del conjunto de resultados completo es igual o menor al umbral, entonces los datos se distribuyen por método broadcast.
- RANGE: Mapea las filas a procesos de consulta utilizando rangos de la clave de ordenación.
- ROUND-ROBIN: Mapea filas a procesos de consulta aleatoriamente.
- BROADCAST: Difunde las filas de la tabla completa a cada proceso de consulta.
- QC(ordenado): El coordinador de la ejecución consume la entrada en orden.

- QC(aleatorio): El coordinador de la ejecución consume la entrada aleatoriamente.

El plan de ejecución de una sentencia paralela almacena, en la columna DISTRIBUTION de PLAN_TABLE, el método usado para distribuir las filas de los procesos productores de consultas a procesos consumidores de consultas.

Después de determinar los requisitos de redistribución para cada operación en el plan de ejecución, el optimizador determina el orden en el que las operaciones deberán ser realizadas. Con esta información, el optimizador determina el plan de paralelismo final y el flujo de datos entre las operaciones paralelas de la sentencia.

En Oracle Database 10g R2, cambia el modelo de ejecución paralela de modelo SQL esclavo a modelo de cursor único paralelo (PSC). En lugar de que el coordinador de la consulta (QC) tenga que construir la sentencia SQL para cada DFO/esclavo en alguna instancia, y que cada esclavo tenga que analizar y ejecutar su propio cursor. A partir de Oracle Database 11gR1, Oracle construye y compila sólo un cursor por instancia que contiene toda la información requerida para la ejecución en paralelo. Por lo tanto, cada esclavo en la misma instancia es capaz de compartir el mismo cursor. Este modelo de plan de paralelismo global mejora tanto las características de rendimiento como la reducción de memoria.

Paralelismo, Oracle Partitioning y Oracle RAC

Por defecto, todas las instancias disponibles de Oracle RAC son consideradas para la ejecución en paralelo. La ejecución en paralelo no asigna los esclavos aleatoriamente a las instancias disponibles, sino que comienza alojando en la instancia menos cargada. El objetivo es tanto minimizar el tráfico inter-node como, al mismo tiempo, intentar minimizar cualquier desequilibrio entre las instancias.

Consultas paralelas y Oracle partitioning

Como se ha mencionado en la sección anterior, la redistribución de operaciones implica el intercambio de filas entre procesos de ejecución paralela. Esta es una operación con una carga elevada de CPU lo que puede producir excesivo tráfico a través del interconnect en entornos Oracle Real Application Clusters.

Uso de Partition-Wise Joins

Cuando un full partition-wise join se ejecuta en paralelo, el granule de paralelismo es una partición. Como resultado, el grado de paralelismo se limita al número de particiones. Por ejemplo, se requieren al menos 16 particiones para establecer un grado de paralelismo de consulta de 16. Se pueden utilizar varios métodos de partitioning para particionar por igual las dos tablas según la columna clave. Partition-wise joins reduce el tiempo de respuesta de una consulta minimizando la cantidad de datos intercambiados entre procesos de ejecución paralela cuando se ejecutan joins en paralelo. Esto reduce significativamente el tiempo de respuesta y mejora el uso tanto los recursos de la CPU como de la memoria. En entornos Oracle Real Application Clusters, lo join de tipo partition-wise también evitan, o al menos

limitan, el tráfico de datos a través de la interconnect, que es la clave para lograr buena escalabilidad para operaciones de join masivas.

La mayoría de las operaciones join en Oracle Real Application Cluster podrían experimentar altas latencias de interconnect sin ejecución paralela de partition-wise joins. Se debe utilizar esta característica para grandes configuraciones DSS que utilizan Oracle Real Application Clusters.

Uso de Grupos de Instancias Paralelas

En un entorno Oracle Real Application Clusters el optimizador tiene en cuenta el coste de enviar un mensaje a través de la interconnect en comparación con enviar el mensaje localmente. También tiene en cuenta el número de instancias activas aunque el optimizador intentará ejecutar la consulta en una sola instancia. Por lo tanto, si se espera que la consulta devuelva un gran número de filas de cada nodo, podría ser beneficioso limitar el paralelismo inter-node como se describió anteriormente y así limitar la cantidad de datos que pasan a través de la interconnect.

Además si cada nodo devuelve sólo un pequeño número de filas podría ser mejor limitar el número de nodos implicados debido a la sobrecarga del tiempo de arranque de procesos remotos. Una forma de minimizar el tráfico inter-node es limitar la ejecución paralela a una instancia o a un grupo de ellas. Para ello, podemos establecer la pertenencia a un grupo a través de los parámetros PARALLEL_INSTANCE_GROUP e INSTANCE_GROUPS

Por ejemplo, considere las siguientes asignaciones:

En la instancia A: INSTANCE_GROUPS=AMER
En la instancia B: INSTANCE_GROUPS=AMER, AMEA, APAC, JPN

Entonces, un usuario puede activar los nodos del grupo AMER para descargar los procesos de consulta utilizando el siguiente comando:

```
ALTER SESSION SET PARALLEL_INSTANCE_GROUP = AMER;
```

Como respuesta, la ejecución paralela puede repartirse entre las instancias, por ejemplo, puede ejecutarse en las instancias A y B. Por otro lado, al poner PARALLEL_INSTANCE_GROUP = APAC, sólo la instancia B puede usarse para la ejecución en paralelo.

Tenga en cuenta sin embargo, que el parámetro init.ora INSTANCE_GROUPS no puede cambiarse dinámicamente.

Nota 1: A partir de Oracle Database 11g la ejecución paralela es consciente de la definición del servicio y automáticamente toma el valor adecuado de PARALLEL_INSTANCE_GROUP, con lo que el establecimiento explícito del valor es innecesario. El uso de servicios se explicará en la sección de gestión de carga de trabajo.

Nota 2: El parámetro INSTANCE_GROUPS ha sido deprecado en 12cR1, se sigue conservando en 12cR2 solo por compatibilidad con versiones anteriores.

El parámetro parallel_adaptive_multi_user (deprecado en 12cR2)

Antes de la introducción de Auto DOP, el paralelismo adaptativo o *Adaptive Parallelism* analizaba la sentencia SQL individual para determinar el DOP ideal de ésta. Las antiguas capacidades del paralelismo adaptativo evaluaban los recursos paralelos para una sentencia en base a la carga de trabajo actual del sistema: Oracle determina en tiempo de ejecución SQL si una operación paralela debe acogerse al DOP solicitado o debe bajar el DOP basándose en la carga de trabajo concurrente del sistema.

En un sistema que hace uso masivo de ejecución paralela usando un DOP alto, el algoritmo adaptativo bajaría el DOP a unas pocas operaciones ejecutándose en paralelo. Mientras que el algoritmo siga garatizando una utilización óptima del sistema, los usuarios podrían experimentar tiempos de respuesta incosistentes. En el peor de los casos, incluso podría hacer que alguna sentencia se ejecutara de forma serializada. Esto da como resultado un rendimiento impredecible para los usuarios, ya que el tiempo de respuesta de una sentencia depende de si está degradado o no.
No es recomendable utilizar capacidades de paralelismo adaptativo en un entorno que requiera tiempos de respuesta deterministas.

El paralelismo adaptativo se controla por el parámetro de inicialización de base de datos PARALLEL_ADAPTIVE_MULTI_USER.

Antes de Oracle Database 12.2, el valor predeterminado de este parámetro era true, lo que significaba que la funcionalidad estaba habilitada de manera predeterminada. Ahora en 12cR2 la funcionalidad ha sido deprecada y el valor predeterminado de este parámetro es false, es decir, que la funcionalidad está deshabilitada de manera predeterminada.

Para controlar la carga y la utilización del sistema, Oracle recomienda el uso de Parallel Statement Queuing y Database Resource Manager. Clasificar a los usuarios con diferentes requisitos de rendimiento en grupos de consumidores dentro del resource manager y asignar recursos paralelos a esos grupos de consumidores en función de los requisitos de rendimiento es una forma mucho mejor de controlar la utilización del sistema y garantizar un rendimiento predecible para los usuarios.

Consideraciones para el interconnect

La mayoría de la demanda de tráfico de interconnect en sistemas de Data Warehouse proviene de la comunicación entre procesos (IPC). Los procesos paralelos se comunican con cada uno de los otros usando la Interconnect. La cantidad de tráfico de la interconnect depende de la operación y del número de nodos que participan en la operación. Las operaciones join tienden a producir más tráfico de interconnect que una simple agregación por la comunicación entre procesos paralelos. La cantidad de tráfico de interconnect puede variar significativamente dependiendo del método de distribución.

El método de distribución usado puede encontrarse en la columna DISTRIBUTION del plan de consulta. Hay casos donde una parte del join es repartida o casos donde dos partes se distribuyen mediante hash dando como resultado la mayoría del tráfico de interconnect. En Partition-wise join parcial en el que sólo una parte del join es redistribuido se produce menor tráfico de interconnect, mientras que en Partition-

wise join completa no se necesita que ninguna parte sea distribuida produciendo aún menor tráfico de interconnect.

Por lo tanto, si se usan consultas paralelas inter-node, la conexión de red debe tener el tamaño apropiado. A menos que la aplicación esté muy bien estructurada con el mayor conjunto de consultas predefinidas dando ventaja a partition-wise joins, debe asegurarse de tener en cuenta qué parte de los datos redistribuidos van a pasar a través de la interconnect para consultas paralelas inter-node.

La cantidad de tráfico de interconnect también depende del número de nodos que participen en la operación join. Cuantos más nodos participen en la operación join, más datos necesitarán ser distribuidos a nodos remotos. Para una instancia en un clúster Oracle RAC de 4 nodos con 4 CPUs cada uno para maximizar el rendimiento de carga con tablas externas se necesita establecer el grado de paralelismo a 32 en las tablas externas e internas. Esto dará como resultado 8 procesos paralelos realizando operaciones de lectura de la tabla externa en cada nodo, así como 8 procesos paralelos realizando sentencias de creación de tablas en cada nodo.

Consideraciones importantes:

Puede usar NIC's con doble o múltiples puertos para redundancia e incrementando el ancho de banda usando agregación de puertos 10 GigE o interconnect Infiniband (si está disponible en su plataforma) si va a usar consultas paralelas inter-node en exceso. Considere el uso del parámetro de inicialización CLUSTER_INTERCONNECTS cuando un único clúster de interconnect no pueda satisfacer sus requisitos de ancho de banda y haya más de una interconnect. Este parámetro le permitirá especificar varias direcciones IP, separadas por comas. El tráfico de red de Oracle RAC se distribuye entre las direcciones IP especificadas.

```
CLUSTER_INTERCONNECTS = ip1:ip2:...:ipn
```

Oracle utiliza todas las interconexiones que se especifiquen. Esto proporciona balanceo de carga siempre y cuando todas las interconexiones de la lista sigan estando operativas. Tenga en cuenta que si una de las interconexiones en el parámetro CLUSTER_INTERCONNECTS llegara a no estar disponible, Oracle devolvería un error y la instancia podría fallar.

Nota: Si usa el parámetro CLUSTER_INTERCONNECTS para balancear la carga y el rendimiento, podría ser a costa de las funcionalidades de alta disponibilidad.

Hay que considerar incrementar el parámetro PARALLEL_EXECUTION_MESSAGE_SIZE. Este parámetro especifica el tamaño del buffer utilizado para la ejecución de mensajes en paralelo. Desde 11.2 el valor por defecto para PARALLEL_EXECUTION_MESSAGE_SIZE es 16Kb, el cual puede ser suficiente para muchos casos. Tenga en cuenta que puede incrementar este valor si usted tiene la memoria libre necesaria en la shared pool.

Monitorice el tráfico de interconnect utilizando informes AWR. Monitorice el tráfico de la caché global (bloques recibidos/enviados, mensajes recibidos/enviados...) y el el tráfico IPQ (mensajes locales y remotos de PX). Resumiendo los mensajes GES+GCS, los mensajes de consulta paralela y los bloques de Cache Fusion, darán una buena estimación del ancho de banda necesario.

Los siguientes son dos ejemplos de informes AWR resaltando los dos principales tipos de tráfico de interconnect:

Tráfico Global Cache (típico para operaciones OLTP)

Global Cache Load Profile	Per Sec	Per Trans
Global Cache blocks received:	2.70	2.23
Global Cache blocks served:	2.84	2.36
GCS/GES messages received:	164.07	136.03
GCS/GES messages sent:	136.96	113.56
DBWR Fusion writes:	0.22	0.18

Tráfico IPQ (para operaciones paralelas)

Statistics	Total	per Sec	per Trans
Mensajes recibidos en PX locales	104	0.1	0.1
Mensajes enviados en PX locales	104	0.1	0.1
Mensajes recibidos en PX remotos	200271	200.2	151.1
Mensajes enviados en PX remotos	213267	213.2	156.1

Carga de datos

Operaciones DML en paralelo

Las operaciones de Lenguaje de Manipulación de Datos (DML) tales como INSERT, UPDATE y DELETE pueden ser paralelizadas por Oracle. La ejecución en paralelo puede acelerar operaciones DML largas y esto es especialmente ventajoso en entornos Data Warehouse donde no es necesario mantener un historial demasiado grande o tablas históricas.

Hay que tener en cuenta que la lectura directa no se utiliza en actualizaciones/borrados paralelos ya que es la caché la que debe actualizar el bloque.

En el siguiente ejemplo, se han generado 12 procesos de ejecución paralela en 3 instancias para esta sentencia UPDATE:

```
SQL> alter session enable parallel dml;
Session altered.
SQL> update /*+ parallel(ware,12) */ ware set w_ytd=w_ytd;
```

El Partitioning permite la ejecución paralela ilimitada de sentencias UPDATE, DELETE, y MERGE. Oracle paralelizará las sentecias SELECT e INSERT cuando ambas accedan a objetos de base de datos tanto particionados como no particionados.

Las sentencias UPDATE, DELETE y MERGE pueden ser paralelizadas tanto para objetos de base de datos particionados como no particionados cuando no hay índices de bitmap, con el fin de paralelizar estas operaciones en objetos que tienen índices de bitmap, la tabla final se debe dividir. La ejecución paralela de estas operaciones SQL puede mejorar el rendimiento en gran medida, especialmente para operaciones UPDATE, DELETE, o MERGE implicadas en grandes volúmenes de datos.

DML paralelo versus paralelismo manual

Las operaciones DML pueden paralelizarse de forma manual con la emisión de varias sentencias DML simultáneas contra diferentes conjuntos de datos. Por ejemplo:

Se puede lanzar múltiples sentencias INSERT en varias instancias de un Oracle Real Application Cluster para hacer uso del espacio libre de varios bloques libres, o emitir varias sentencias UPDATE y DELETE con diferentes rangos de valor de clave o rangos de id columna.

Sin embargo, el paralelismo manual tiene los siguientes inconvenientes:

- Es difícil de usar. Usted tiene que abrir varias sesiones (posiblemente en instancias diferentes) y emitir varias sentencias.
- Hay una falta de propiedades transaccionales. Las sentencias DML se emiten en momentos diferentes; y, en consecuencia, los cambios se hacen dando lugar a versiones inconsistentes de la base de datos. Para obtener atomicidad, el commit o rollback de varias sentencias debe coordinarse manualmente (tal vez a través de instancias).
- La división del trabajo es compleja. Usted tendrá que consultar la tabla en orden para encontrar el id de columna o el rango de valor de clave idóneos para dividir correctamente el trabajo.
- El cálculo es complejo. El cálculo del grado de paralelismo puede ser complejo.
- Existe una falta de afinidad y de recursos de información. Usted necesita conocer información certera para emitir la sentencia DML correcta en la instancia correcta cuando corre Oracle Real Application Cluster. Usted también tiene que conocer el uso de recursos para balancear la carga entre las distintas instancias.

La ejecución de operaciones DML en paralelo elimina estos inconvenientes realizando inserts, updates y deletes en paralelo automáticamente.

Contención de tipo Index Block

En sistemas Data Warehouse, donde la carga o el procesamiento de de datos por lotes es la función de negocio predominante, puede haber problemas de rendimiento que afectan al tiempo de respuesta debido a la gran cantidad de inserciones en índices. Dependiendo de la frecuencia de acceso y del número de procesos concurrentes insertando o actualizando datos, los índices pueden llegar a ser puntos conflictivos (o como se denominan en inglés *hot*) y la contención puede venir provocada por:

- Índices basados en secuencias ordenadas monótonamente crecientes,
- Frecuentes divisiones de las hojas/bloques de los índices,
- Baja profundidad de los índices, donde todos los bloques se accedan desde el nodo raíz.

Como recomendación general, para aliviar el impacto en el rendimiento de los bloques de índice calientes globales (*globally hot index blocks*) y las divisiones de bloques (*leaf block splits*), una distribución más uniforme y menos sesgada de la concurrencia en el árbol de índice debería ser el objetivo principal. Esto se puede lograr por:

1. *Partitioning de tipo Global Index Hash*

   En Oracle Database 10g Release 2, un índice global puede ser particionado mediante hash. Esto reducirá la contienda por índices bloqueados cuando se inserta frecuentemente en segmentos con índices que posiblemente crecen por la derecha. En un entorno data warehouse, también podría ser necesario un índice global cuando su consulta no pueda mejorar por la forma en la que se particionó la tabla.

   Por ejemplo, algunas consultas necesitan recorrer todo el índice para recuperar información. Si el índice es LOCAL, Oracle ejecutará y combinará los datos de todas las particiones del índice dando como resultado un bajo rendimiento. Si el índice ha sido creado GLOBAL, Oracle realizaría menos lecturas de un único índice (pero más grande).

   Con el índice global se obtendrá mejor rendimiento y se aplicará el paralelismo si se particiona mediante hash por la clave de índice. En el esquema de la partición del índice global, el índice es más difícil de mantener ya que puede abarcar particiones en la tabla base. Por ejemplo, cuando falla una partición de una tabla que forma parte de una reorganización, el índice global completo quedará invalidado y deberá ser reconstruido usando:

   ```
   alter table <tablename> drop <partition name> update global indexes;
   ```

2. *Aumentar la caché de la secuencia, si el valor de clave se deriva de una secuencia*

3. *Uso de natural keys en lugar de surrogate keys*

4. *Uso de índices de tipo Reverse Key*

   Un índice de clave invertida funciona invirtiendo el orden de los bytes del valor de la clave. Invertir las claves de los índices permite que las inserciones se distribuyan a través de todas las claves hoja en el índice. Estos índices son diseñados para eliminar los puntos conflictivos en solicitudes de inserción y son excelentes para mejorar el rendimiento de las inserciones. Sin embargo, una de las mayores limitaciones de los índices de clave invertida es que no pueden usarse como índice de recorrido por rango, ya que al invertir la clave del índice aleatoriamente se distribuyen los bloques por los nodos de índice hoja. Un índice con clave invertida sólo puede usar los métodos de acceso fetch-by-key y full-index(table)scan.

5. *Uso de bloques de diferente tamaño*

   El uso de tamaños de bloque mayores en Oracle suelen dar menos niveles de índices y por lo tanto, se mejora el tiempo de acceso de índices a datos. Una simple I/O obtendrá muchas filas relacionadas y las siguientes solicitudes de las siguientes columnas estarán ya en el buffer de datos. Esta es una de las principales ventajas de un tamaño de bloque mayor. Otra ventaja es que se reducirá el número de divisiones.

   Si se usa un tamaño de bloque menor, la reducción en el número de filas en un bloque ayudará a reducir la contención por buffer busy en caso de alta

concurrencia. En general, esto sería más bien un problema para los procesos update o merge, más que para insert masivos. Así, para un sistema DW en el que las inserciones predominan sobre las actividades DML, el uso de bloques más pequeños no ofrece ningún beneficio.

Existen también algunos casos en los que también puede resultar beneficioso el uso de una única instancia para el proceso de carga. Este enfoque se discutirá en la siguiente sección.

Automatic Workload Management

La gestión automática de la carga de trabajo (AWM) facilita y controla la distribución del trabajo a través de los nodos en el clúster con el fin de lograr un rendimiento óptimo para usuarios y aplicaciones.

Servicios

Desde Oracle Database 10g se incorpora la posibilidad de gestionar la carga de trabajo mediante la definición de servicios. Son la base para la gestión de la carga de trabajo en Oracle RAC. Los servicios dividen la carga de trabajo que se ejecuta en toda la base de datos Oracle en clases disjuntas. Cada servicio representa una carga de trabajo/aplicación con atributos comunes, umbrales de nivel de servicio, prioridades, medidas de rendimiento para transacciones reales y alertas y acciones para cuando los objetivos de rendimiento han sido incumplidos.

Database Services y Global Data Services

Un servicio de base de datos representa una sola base de datos. Esta base de datos puede ser una base de datos single-instance o una base de datos de Oracle Real Application Clusters (Oracle RAC) con múltiples instancias concurrentes de la misma base de datos. Un Global Data Services es un servicio proporcionado por múltiples bases de datos sincronizadas a través de replicación de datos (Active Data Guard, Golden Gate...).

Comenzando con Oracle Database 12c, puede usar Global Data Services (GDS) para la administración de carga de trabajo que involucra múltiples bases de datos Oracle. GDS permite a los administradores gestionar de forma automática y transparente las cargas de trabajo de los clientes en las bases de datos replicadas que ofrecen servicios comunes. Estos servicios comunes se conocen como servicios globales.

Cuando se crea un servicio, se definen las instancias que normalmente soportarán ese servicio. También se pueden definir otras instancias para soportar un servicio por si la instancia preferida falla. Éstas se conocen como instancias disponibles para un servicio.

Cuando se especifica una instancia preferida para un servicio, el servicio se ejecuta en esta instancia durante una operación normal. Oracle Clusterware intenta garantizar que el servicio siempre se ejecutará en las instancias preferidas que han sido configuradas para este servicio. Si la instancia falla, el servicio es realojado aleatoriamente en una de las instancias disponibles. Si no se especifican instancias

preferidas o disponibles cuando se crea un servicio, por defecto todas las instancias de Oracle RAC database serán instancias preferidas para este servicio.

Un servicio puede definirse usando srvctl u Oracle Enterprise manager-Grid Control. Por ejemplo:

```
srvctl add service -d rac_db -s dw_load -r inst1,inst2 -a inst3,inst4
srvctl add service -d rac_db -s dw_query -r inst3,inst4 -a inst1,inst2
```

En este ejemplo, se crean dos servicios, DW_LOAD y DW_QUERY. El servicio DW_LOAD usa las instancias inst1 e inst2 como sus instancias preferidas, e inst3 e inst4 como sus instancias disponibles. El servicio DW_QUERY usa estas asignaciones a la inversa.

Las aplicaciones middle tier y las aplicaciones cliente-servidor utilizan un servicio especificando ese servicio como parte de una conexión en el TNS connect data. Puede encontrarse en el archivo TNSnames para los Net drivers, en la URL de especificación para drivers o puede mantenerse en Oracle Internet Directory.

Ejemplos de tnsnames.ora:

- Sin SCAN

```
LOAD_USERS =
 (DESCRIPTION =
  (ADDRESS_LIST = Service
   (ADDRESS = (PROTOCOL = TCP)(HOST = docrac1-vip)(PORT = 1521))
   (ADDRESS = (PROTOCOL = TCP)(HOST = docrac2-vip)(PORT = 1521))
   (LOAD_BALANCE = yes)
  )
  (CONNECT_DATA = (SERVICE_NAME = DW_LOAD))
 )
```

- Con SCAN

```
LOAD_USERS =
 (DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = scan-name-rac.oracle.com)(PORT = 1521))
  (CONNECT_DATA = (SERVICE_NAME = DW_LOAD))
 )
```

En el lado servidor, el Job Scheduler, Parallel Query u Oracle Advanced Queuing ponen el nombre del servicio como parte de la definición de la carga de trabajo.

Para el Job Scheduler, el servicio que una clase de tarea usa se define cuando la clase de tarea se crea. Durante la ejecución, las tareas se asignan a las clases de tareas y las clases de tareas se ejecutan como servicios. El uso de servicios con clases de tareas garantiza que el funcionamiento del planificador de tareas se identifique para la gestión de la carga de trabajo y la optimización del rendimiento. Para conseguir alta disponibilidad DBMS_JOBS previamente usa el nombre de la instancia para definir dónde deben ser ejecutados los servicios. Este planteamiento dio lugar a que hubiera tareas que no se ejecutaban cuando la instancia no estaba disponible. Con el nuevo planificador de tareas Job Scheduler, poner el servicio en la clase de tarea asegura que la tarea se ejecuta cuando el servicio se ejecuta en algún lugar del cluster.

Servicios y consultas en paralelo

Para consultas y DML en paralelo, el coordinador de la consulta se conecta a un servicio como cualquier otro cliente. Sin embargo, en Oracle RAC 10g, la ejecución paralela de esclavos que han sido asignados a instancias sin tener en cuenta los servicios reduce en gran medida los beneficios de los servicios. La solución consiste en usar parallel_instance_groups. Desde Oracle RAC 11g, la ejecución paralela se ha integrado con los servicios y los esclavos se restringen automáticamente a instancias donde el servicio se está ejecutando, haciendo uso innecesario del parámetro parallel_instance_group.

Limitar el número de recursos para un usuario

La cantidad de paralelismo disponible para un usuario dado puede limitarse estableciendo un grupo consumidor de recursos para el usuario. Con ello, podemos limitar el número de sesiones, logins concurrentes, o el número de procesos paralelos que algún usuario o grupo de usuarios pueden tener.

Cada proceso de consulta que trabaja sobre una sentencia de ejecución en paralelo se inicia con un ID de sesión. Cada proceso cuenta el límite de sesiones concurrentes de usuario. Por ejemplo, para limitar un usuario a 10 procesos en ejecución en paralelo, se establece el límite de usuario a 11. Un proceso es para el coordinador paralelo y los otros 10 forman dos conjuntos de procesos de consulta. Esto daría lugar a una sesión para el coordinador paralelo y 10 sesiones para los procesos de ejecución paralela.

Implementación de servicios

Podemos usar los servicios para gestionar y dividir la carga de trabajo: Los servicios proporcionan una única imagen del sistema para gestionar la carga de trabajo de servicios que abarcan una o más instancias de la base de datos. Una instancia puede soportar varios servicios. El número de instancias que ofrecen servicios es administrado por el DBA e independientemente de la aplicación define un servicio para aplicaciones que tienen el mismo nivel de cumplimiento y requisitos similares.

En base al SLA de cada servicio, se decide cuántas instancias ofrecerán un servicio dado. Por ejemplo, defina un servicio que ejecute procesos de carga, un segundo servicio para consultas ad-hoc y un tercer servicio para operaciones por lotes. En base al sistema y a la disponibilidad de recursos usted decidirá qué instancias ofrecerán cada uno de los servicios, comprobando si hay servicios que deberían ejecutarse en una instancia por razones de rendimiento (contienda por índices durante el proceso de carga, por ejemplo).

Consideraciones adicionales

Filtros Bloom

Desde Oracle Database 11g, el rendimiento de partition pruning se ha mejorado con el uso de filtros de tipo bloom en lugar de usar subquery pruning. Mientras que para

subquery pruning se necesitaba decidir su activación suponiendo un coste y un consumo de recursos internos (recursivos), el pruning basado en filtros bloom está activo todo el tiempo sin consumir recursos adicionales.

Un filtro Bloom es un algoritmo probabilístico que rápidamente comprueba la pertenencia a un conjunto grande con el uso de varias funciones hash en un único array de bits. Puede usarse un filtro de join cuando un plan de ejecución recupera todas las filas de una tabla antes de recuperar algunas filas de la otra tabla con la que se hace el join:

- El filtro join construye un array de bits, y convierte a bits cada fila de la primera tabla que coincida con las condiciones de búsqueda.

- Cuando se recorre la segunda tabla, las filas a las que no ha sido posible hacer el join con la primera tabla se rechazan.

Como ya se ha mencionado, en un join hash paralelo con paralelismo inter-node, la interconnect puede llegar a ser un cuello de botella. Haciendo un prefiltrado usando filtros bloom se puede reducir el overhead de la cominicación.

Ejemplo de plan de ejecución con el filtro bloom:

```
select count(*)
from empp e, deptp d
where e.deptno = d.deptno
```

```
SELECT STATEMENT            |             |            |      |
     SORT AGGREGATE         |             |            |      |
      PX COORDINATOR        |             |            |      |
       PX SEND QC (RANDOM)  | :TQ10002    |         02 | P->S | QC
        SORT AGGREGATE      |             |         02 | PCWP |
         HASH JOIN          |             |         02 | PCWP |
          PX JOIN FILTER CREATE| :BF0000   |         02 | PCWP |
           PX RECEIVE       |             |         02 | PCWP |
            PX SEND HASH    | :TQ10000    |         00 | P->P | HASH
             PX BLOCK ITERATOR |          |         00 | PCWC |
              TABLE ACCESS FULL| EMPP      |         00 | PCWP |
          PX RECEIVE        |             |         02 | PCWP |
           PX SEND HASH     | :TQ10001    |         01 | P->P | HASH
            PX JOIN FILTER USE | :BF0000   |         01 | PCWP |
             PX BLOCK ITERATOR |          |         01 | PCWC |
              TABLE ACCESS FULL| DEPTP     |         01 | PCWP |
```

Ejemplo de un plan de ejecución con un filtro bloom pruning:

```
| Id  | Operation                | Name     | Pstart| Pstop |
-------------------------------------------------------------
|   0 | SELECT STATEMENT         |          |       |       |
|   1 |  SORT AGGREGATE          |          |       |       |
|*  2 |   HASH JOIN              |          |       |       |
|   3 |    PART JOIN FILTER CREATE | :BF0000 |       |       |
|   4 |     PARTITION HASH ALL   |          |     1 |     2 |
|   5 |      TABLE ACCESS FULL   | TB1      |     1 |     2 |
|   6 |    PARTITION HASH JOIN-FILTER|      |:BF0000|:BF0000|
|   7 |     TABLE ACCESS FULL    | TB2      |:BF0000|:BF0000|
```

Desde Oracle 11g el filtro bloom está habilitado por defecto.

Consderaciones para tablespace temporal

El tablespace temporal es un recurso global. Cada instancia en un Oracle RAC, como utiliza el espacio temporal 'soft' se reserva el espacio en su SGA privado. El espacio de temporal, una vez alojado en una instancia, no es visible para otras instancias y las instancias no pueden devolver el espacio temporal al pool común. Así que cuando un proceso se queda sin espacio para asignar en el tablespace temporal, solicita a otras instancias en el clúster que liberen el espacio no utilizado. Esto se hace por medio de Cross Instance Call (CI Call).

La sesión que solicita el espacio obtendrá el 'SS enqueue' para la tablespace temporal y emitirá una cross instance call (usando un CI enqueue) hacia las otras instancias (esperando por 'DFS lock handle'). Todo el espacio temporal solicitado a las instancias se serializará en su cola 'CI', y esto puede tener un coste muy alto si la aplicación usa mucho espacio temporal para tablas temporales, segmentos ordenados, etc. (muchas sesiones esperando por 'SS enqueue' y 'DFS lock handle') y puede causar problemas graves de rendimiento.

Las Best Practices para alojar una tablespace temporal son:

- Asegurarse de que se configura el espacio temporal suficiente. Debido a la forma en que una instancia de Oracle RAC gestiona el espacio temporal, podría ser útil asignar espacio adicional en comparación con una base de datos de single-instance.

- Aísle usuarios con mucho uso o un uso muy variable de espacio temporal para separar tablespaces temporales. Separar usuarios que sacan informes de usuarios OLTP podría ser una opción.

- Monitorizar la asignación de espacio temporal para asegurarse de que cada instancia tiene suficiente espacio temporal disponible y que el espacio temporal se asigna por igual entre las instancias. Se pueden usar las siguientes sentencias SQL:

```
select inst_id, tablespace_name, segment_file, total_blocks, used_blocks,
free_blocks, max_used_blocks, max_sort_blocks
from gv$sort_segment;

select inst_id, tablespace_name, blocks_cached, blocks_used
from gv$temp_extent_pool;

select inst_id,tablespace_name, blocks_used, blocks_free
from gv$temp_space_header;

select inst_id,free_requests,freed_extents
from gv$sort_segment;
```

  Si la asignación de espacio temporal entre instancias no hubiera sido balanceada, podría ser necesario dar de baja manualmente los segmentos temporales de una instancia. Para esto se usan los siguientes comandos:

```
"alter session set events 'immediate trace name drop_segments level <TS
number + 1>;"
```

- Para cada tablespace temporal, se asignan al menos tantos ficheros temporales como instancias haya en el clúster. Ver notas Metalink 465840.1 y 469036.1 para más información al respecto.

Cuándo aplicar estas best practices

Las best practices presentadas anteriormente se aplican a todos los entornos Data Warehouse en Oracle RAC. ¿Cómo se deben aplicar a su sistema data warehouse? Esto depende. En última instancia, las best practices para implementar Data Warehouse en Oracle RAC dependerán de muchas variables que deben incluirse en una lista definitiva que se deberá seguir. ¡Tiene que conocer a fondo de todas ellas, para que pueda aplicar sólo aquellas que son adecuadas para su entorno y carga de trabajo y que satisfacen sus necesidades empresariales. Es importante:

- Entender y planificar el tráfico de interconnect

  La diferencia entre Data Warehouse con Oracle RAC y sin Oracle RAC es el uso de la interconnect en operaciones paralelas. Comprenda que el tráfico de interconnect le ayudará a decidir qué ancho de banda de interconnect es necesario y cómo manejarlo. Además de esto, las consideraciones de alta disponibilidad necesitan ser tenidas en cuenta cuando se planifique la configuración de interconnect.

- Configurar Consultas Paralelas y Operaciones Paralelas

  Las operaciones en paralelo dentro de un entorno Oracle RAC dan la flexibilidad para utilizar todos los servidores que forman parte del clúster. En algunos casos será mejor usar todos los recursos disponibles, mientras que en otros casos será mejor restringir las operaciones en paralelo a un sólo nodo. El grado de paralelismo puede definirse para utilizar todos los recursos disponibles en el clúster, o para restringir las operaciones en paralelo a uno o a un conjunto de nodos del clúster. También mediante el uso de grupos de instancias y servicios, usted puede controlar la asignación de recursos en base a su implementación, los requisitos de aplicación o de nivel de servicio. Así, los nodos pueden agruparse de forma lógica para tipos específicos de operaciones.

- Gestionar su carga de trabajo

  Usted debería gestionar su carga de trabajo usando Servicios. Tendrá que decidir cuántos servicios definir y cuántas instancias ofrecerán un determinado servicio en un momento concreto. Por ejemplo, en algunos casos será mejor ejecutar un proceso de carga en una única instancia para evitar la contienda.

  Además, en algunos casos el gestor de recursos debería usarse para asignar de forma óptima los recursos a los servicios.

- Encontrar la estrategia óptima para Partitioning

  Las particiones son la base para lograr un buen rendimiento en Data Warehouse de gran tamaño y otras características también dependen del partitioning para conseguir sus objetivos. Los dos criterios más importantes a tener en cuenta cuando se elige partitioning son el rendimiento y la facilidad de administración. Una estrategia de partitioning determinada debería llevarse a cabo por consideraciones de rendimiento. Esto significa que se optimizarán las consultas de usuario, informes, descargas, etc. para conseguir mejor rendimiento y

escalabilidad, a menudo a costa del rendimiento operacional y de la facilidad de uso.

A modo de ejemplo, los datos de ventas a menudo pueden ser analizados por su ubicación geográfica primero y después por fecha (por ejemplo, ventas por región durante el mes pasado), mientras que los procesos ETL probablemente cargarán los datos en base a una fecha (por ejemplo, las ventas de esta semana se cargan en un único lote de cada región geográfica). El conflicto en este ejemplo es que los datos pueden particionarse primero o por región geográfica o por fecha. Uno de los enfoques favorecerá los accesos de usuario, mientras que el otro optimizará las cargas ETL.

Por último, se debe conseguir un equilibrio, aunque principalmente se dará soporte a los usuarios.

En una implementación Oracle RAC, esto es aún más importante que el esfuerzo invertido en reducir el tráfico de contención de tipo inter-node. Para ello, la carga de trabajo se particiona de la misma forma que los datos. Esto reducirá la probabilidad de grandes cantidades de datos que van y vienen entre los nodos del clúster. Un ejemplo, basado en los datos de ventas ya mencionados, sería que los informes de usuarios para una región geográfica en particular pudieran procesarse en nodos específicos, mientras que otros pudieran ser gestionados en otros nodos – así, los datos procedentes de ciertas particiones serían accedidas exclusivamente por nodos particulares, reduciendo enormemente el tráfico de interconnect.

- Medir y Monitorizar Continuamente

  En un sistema Data Warehouse la carga cambia diariamente, en cada hora, incluso minuto a minuto. Es imprescindible que el trabajo realizado, el uso de recursos, el rendimiento del disco, la actividad de red y en general la salud del sistema Oracle RAC sean monitorizados y gestionados activamente. Tal enfoque tan exigente para el cuidado del sistema Data Warehouse permitirá un uso más óptimo del sistema de soporte de Oracle RAC y de sus recursos.

- Diseñar y probar para satisfacer las necesidades específicas del negocio

  Un entorno común para sistemas data warehouse, podrían ser todos los sistemas informáticos. Sin embargo, tiene una importancia aún mayor en un sistema de de tipo dataware house. Esto se debe a los sistemas de almacenamiento de datos por lo general han estado limitados por el uso de requisitos específicos. Ahora, que estos sistemas han evolucionado y han asumido más de un perfil mixto de carga, especialmente con las necesidades empresariales en constante cambio, cada vez es más difícil diseñar con eficacia.

  Este estado de cambio que caracteriza a muchos de los sistemas Data Warehouse más recientes se ha convertido en una clave de negocio "necesaria". Por lo tanto, el diseño de un sistema Data Warehouse debe impregnarse de la flexibilidad necesaria para satisfacer requisitos cambiantes.

  Un sistema Oracle RAC, por su propia naturaleza, es adaptable, lo que le permite satisfacer mejor las necesidades de desarrollo del sistema data warehouse.

Para confirmar que realmente ha diseñado correctamente su sistema Data Warehouse en Oracle RAC, para satisfacer las demandas del negocio y el rendimiento, debe probarlo - a fondo y bajo una carga adecuada, con objetivos funcionales y de rendimiento claramente definidos.

Conclusiones

Como los sistemas Data Warehouse modernos y sus requisitos técnicos se han vuelto más comunes, y más complejos, éstos exigen mayor flexibilidad, mayor disponibilidad y un mejor rendimiento de su infraestructura de base de datos. Estas crecientes demandas se pueden atender fácilmente con la implementación de Oracle Real Application Clusters (RAC). Esta arquitectura, combinada con otras características de RDBMS de Oracle, puede ofrecer la estabilidad, disponibilidad, escalabilidad y funcionalidad en general para que el sistema Data Warehouse sea un éxito.

Además, el éxito del sistema Data Warehouse implementado en Oracle RAC depende del equilibrio en el diseño entre las optimizaciones para las necesidades del negocio frente a las necesidades operativas, el equilibrio en la implementación para soportar el acceso concurrente de unos cuantos usuarios en comparación con el rendimiento escalable para adaptarse a cualquier número de demandas de procesamiento simultáneo. En última instancia, el sistema debe ser diseñado e implementado para satisfacer mejor las necesidades de la empresa y al mismo tiempo, mantener las exigencias de funcionamiento al mínimo.