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 de desarrollo sobre Oracle Real Application Cluster

Referencia documento:
InfV5_JASAS_RAC_Development_BestPractices_V920.doc
Fecha: 16 de noviembre de 2018
Versión: 9.2.0

Registro de Cambios

Fecha	Autor	Versión	Notas
14 de Octubre de 2010	Oracle ACS	1.4.0	Versión inicial
13 de Enero de 2011	Oracle ACS	2.1.0	Versión 2.1
14 de Abril de 2011	Oracle ACS	2.2.0	Versión 2.2
17 de octubre de 2013	Oracle ACS	2.3.0	Versión 2.3
13 de Octubre de 2011	Oracle ACS	2.4.0	Versión 2.4
12 de Enero de 2012	Oracle ACS	3.1.0	Versión 3.1
14 de Marzo de 2	Oracle ACS	4.1.0	Revisión de Marzo de 2013, contrato 2012-2014
13 de Junio de 2013	Oracle ACS	4.2.0	Revisión de Junio de 2013, contrato 2012-2014
17 de Octubre de 2013	Oracle ACS	4.3.0	Revisión de Octubre de 2013, contrato 2012-2014
16 de Julio de 2015	Enrique Ramiro	6.1.0	Revisión de Julio de 2015, contrato 2014-2016
16 de Diciembre de 2015	Enrique Ramiro	6.2.0	Revisión de Diciembre de 2015, contrato 2014-2016
16 de Junio de 2016	Enrique Ramiro	7.1.0	Revisión de Junio de 2016, contrato 2014-2016
16 de Noviembre de 2016	Enrique Ramiro	7.2.0	Revisión de Noviembre de 2016, contrato 2014-2016
16 de Junio de 2017	Enrique Ramiro	8.1.0	Revisión de Junio de 2017, contrato 2016-2018
16 de Noviembre de 2017	Enrique Ramiro	8.2.0	Revisión de Noviembre de 2017, contrato 2016-2018
16 de Junio de 2.018	Enrique Ramiro	9.1.0	Revisión de Junio de 2018, contrato 2016-2018
16 de Noviembre de 2018	Enrique Ramiro	9.2.0	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 aplicaciones 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 112cR2 Real Application Cluster (RAC), 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 el desarrollo de aplicaciones 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, aunque algunas de las recomendaciones son igualmente aplicables a las versiones 9iR2, 10gR2, 11gR1 y 11gR2.

El objetivo de esta guía de buenas prácticas tiene varios objetivos:

- Aprovechamiento de las características del producto Oracle RDBMS Real Application Cluster.
    - Escalabilidad
    - Alta disponibilidad
    - Balanceo de carga
- Facilitar la implantación y modificación del aplicativo en producción con sistemas basados en Oracle RDBMS Real Application Cluster.

Best practices de desarrollo de aplicaciones sobre RAC

Automatic Segment Space Management (ASSM)

La gestión del espacio en una base de datos es una de las tareas más importantes de los administradores. A esta tarea suelen dedicar bastante tiempo de su trabajo, donde además de la gestión, realizan planificación y monitorizaciones de la utilización de espacio de almacenamiento con el fin de cumplir tanto los requisitos de disponibilidad como los de ahorro de costes.

Una de las nuevas funcionalidades introducidas en Oracle 9i y que simplificó todas las tares de administración de espacio, implantó el uso de best practices y sobre todo eliminó la dependencia entre rendimiento y gestión de espacio fue Automatic Segment Space Management (ASSM).

ASSM puede describirse como una única funcionalidad que simplifica la gestión del espacio libre tanto de tablas como de índices, mejora la utilización del espacio y provee de un significativo aumento del rendimiento y de la escalabilidad de una base de datos.

En versión anteriores a Oracle 9i, las estructuras de datos llamadas FREELISTS almacenaban la información necesaria sobre un objeto para saber si existía o no espacio suficiente para realizar la inserción de una nueva fila. Los administradores podían definir el número de FREELIST y de FREELIST GROUPS durante la creación del objeto en cuestión.

ASSM provee de un nuevo mecanismo que hace la gestión del espacio de un objeto totalmente transparente al usar bitmaps para gestión el espacio de cada bloque de datos de un objeto. El estado del bitmap indica cuando espacio libre existe para un bloque dado, por ejemplo, por encima del 75%, entre el 50% y el 75%, entre el 25% y 50% o menos del 25%, además de indicar si está o no formateado.

Esta nueva implementación elimina la necesidad de ajustar los parametros relacionados con la gestión de espacio, como FREELISTS, FREELIST GROUPS y PCTUSED, liberando al administrador de la gestión manual del espacio de los objetos de una base de datos, a la misma vez que mejora la utilización del espacio, permite un mejor conocimiento de cuando espacio libre existe realmente, especialmente en aquellos objetos con un gran variabilidad el tamaño de sus filas.

Adicionalmente, ASSM mejora el rendimiento de las operaciones DML concurrente ya que los bitmap de ASSM permite el acceso simultaneo eliminando cualquier serialización en la búsqueda de espacio libre.

Ventajas de ASSM en entornos RAC

El rendimiento y la manejabilidad que permite ASSM son particularmente interesantes en entornos basadas en RAC, ya que elimina la necesidad de modificar el número de FREELISTS y FREELISTS GROUPS cuando se añaden nuevas instancias al clúster evitando así cualquier tipo de indisponibilidad derivadas de tareas de reorganización.

En pruebas internas se ha comprobado un aumento de hasta un 35% en el número de filas insertadas de forma masiva sobre una tabla gestionada por ASM frente a su equivalente gestionada manualmente, concretamente, 8 FREELIST GROUPS y 20 FREELISTS.

Uso de Automatic Segment Space Management

ASSM está exclusivamente disponible sobre tablespaces manejados localmente, locally managed. La clausula SEGMENT SPACE MANAGEMENT en la operación de CREATE TABLESPACE permite al administrador elegir entre los modos automático y manual, ya que si un tablespace es creado con la clausula MANUAL continuará usando la gestión de espacio libre basada en FREELISTS.

La funcionalidad de ASSM está ligada al parámetro COMPATIBLE que debe tener al menos el valor 9.0.0.

Ejemplo de uso de ASSM durante la creación de una tablespace:

```
CREATE TABLESPACE data DATAFILE
'/u02/oracle/data/data01.dbf' SIZE 50M EXTENT MANAGEMENT
LOCAL SEGMENT SPACE MANAGEMENT AUTO;
```

En el caso del ejemplo, cualquier objeto creado sobre el tablespace usará ASSM para la gestión del espacio libre. Cualquier expecificación de los cluasulas PCTUSED, FREELISTS y FREELIST GROUPS serán ignoradas

Finalmente, se ha añadido un nueva columna a la vista DBA_TABLESPACES, concretamente la columna SEGMENT_SPACE_MANAGEMENT que indica el método de gestión de espacio del tablespace.

Particionamiento

El particionamiento de objetos, tanto tablas como índices es una técnica común y ampliamente aceptada en entornos single-instance ya que permite el aumento de la concurrencia en el acceso a los datos a consta de separar lógicamente la información con el fin de minimizar la cantidad de datos a tratar.

Esta ventaja se convierte en un factor decisivo en el caso de entornos RAC, donde la capacidad de concurrencia debe extenderse a un sistema multi instancia y donde se espera un crecimiento cercano al lineal del sistema.

En este tipo de entornos, hay que tener en cuenta además, el hándicap debido a los problemas inherentes del acceso concurrente a un recurso compartido y la actividad de las instancias a la hora de organizarse para compartir dicha información.

Por tanto, el objetivo es doble, disminuir la contención, entendida como la posibilidad de que dos o más instancias necesiten acceder a un mismo recurso en el mismo momento del tiempo y disminuir las necesidades de comunicación entre las instancias.

Existen tres tipos básicos de particionamiento ó *Single-Level Partitioning*:

- Particionamiento por rango

- Particionamiento por lista
- Particionamiento por función HASH.

Particionamiento por rango o lista

El particionamiento por Rango o por Lista puede ser muy efectivo si la carga de trabajo puede ser orientada a una determinada instancia, en función de los valores de los datos. Esto significa que, dirigiendo determinados módulos de la aplicación o programas concretos a instancias particulares, se puede reducir la frecuencia de las transferencias entre caches.

Particionamiento por función HASH.

El particionamiento por funciones HASH tiene como principal objetivo reducir la contención de acceso sobre bloques alojados en memoria, eventos de espera BUFFER BUSY, mediante la dispersión de las inserciones en distintas particiones. Tiene un efecto similar al uso de de índices inversos, que veremos en un posterior apartado, pero sin la limitación de estos en cuanto a lo no equidad.

En Oracle RDBMS 9iR1, para particionar un índice usando una partición HASH, la única forma de realizarlo era particionando igualmente la tabla correspondiente a dicho índice. A partir de la versión 10g, es posible particionar tan solo el índice. Esto permite una implementación más sencilla en los casos donde la contención es claramente sobre bloques correspondientes a los índices.

Los índices globales con particiones hash pueden mejorar el rendimiento de los índices donde una pequeña cantidad de leaf blocks en el índice tienen una alta contención en entornos OLTP de múltiples usuarios.

En ambos sistemas de particionamiento, es fundamental tomar una serie de decisiones que a la postre, influirán en el aprovechamiento de las técnicas de particionamiento y por tanto, en el rendimiento general del sistema.

La elección de la clave de partición o partition key es el primer paso a realizar y consiste en la elección de la columna o columnas que se usarán para la generación del particionamiento. Es importante recordar, que estas columnas no tienen por qué coincidir con las columnas un índice en concreto, en el caso de que deseemos eliminar la contención sobre él.

En este punto, existen dos factores que debe dirigir la elección de la partition key:

- Eliminación de particiones, cualquier consulta que contenga un predicado de igualdad sobre la clave de partición exclusivamente usará una única partición de índice. El resto de las consultas, así como las que usen otros predicados sobre otros índices, deberán usar todas las particiones del índice, lo que conllevará un aumento en las lecturas lógicas (logical reads). Por esta razón, la clave de partición debe ser frecuentemente usada en los predicados de las consultas con el fin de eliminar la mayor cantidad de particiones.

- Heterogeneidad de valores, la clave de partición debe tener un número considerable de valores diferentes, con el fin de que la función HASH consiga una dispersión mayor y homogénea.

El número de particiones viene determinado por los siguientes factores:

- Los algoritmos HASH tiene una mejor distribución de los datos si el número de particiones es potencia de 2.

- En tests de carga se ha observado una gran reducción de la contención hasta un número de 64 particiones. Por encima de este valor, el grado contención disminuye pero en menor grado.

  - Sin embargo, estas afirmaciones pueden variar en función de la concurrencia y el tipo de la carga que soporta la base de datos, especialmente en proceso concurrentes de inserción de datos.

  - En Oracle RDBMS 9iR2, la memoria destinada a la SHARED POOL suele incrementarse proporcionalmente al uso de particiones y al número de cursores que hacen uso de ellas. Esta afirmación no aplica en versiones Oracle 10g y superiores.

- Finalmente, el número de conjuntos de búferes usados en sentencias de tipo INSERT también se incrementa proporcional al número de particiones, por lo que puede esperarse un aumento de la memoria destinada a la BUFFER CACHE.

En cuanto al particionamiento de índices, existen dos opciones, el particionamiento LOCAL o el particionamiento GLOBAL de éstos.

- En el caso de particionamiento de una tabla con múltiples índices, uno de ellos debe ser el que decida la configuración del conjunto de ellos.

- El índice que motive o dirija el particionamiento debe ser creado como LOCAL, para que las operaciones de tipo INSERT se realicen a lo largo de las diferentes particiones. Para el resto de los índices, la recomendación es crearlos como GLOBAL para que no sufran contenciones de tipo BUFFER BUSY además de prevenir que las sentencias SQL realicen escaneo de múltiples particiones.

Ventajas

Las ventajas del particionamiento de tablas e índices se muestran a continuación:

- Reduce la contención por BUFFER BUSY al permitir que las sentencias de tipo INSERT se repartan a través de las diferentes particiones tanto de las tablas como de los índices.

- Ayudan a la reducción de la contención por BUFFER BUSY y el uso de latches de tipo CACHE BUFFER CHAINS incluso en entorno mono-instancia.

- No poseen la limitación de los índices reversos, ya que puede usarse los índices tanto para no-igualdades como para igualdades en la cláusula WHERE de la sentencias SQL y DML.

Desventajas

En cuanto a las desventajas del particionamiento de tablas e índices se muestran a continuación:

- Exclusivamente las sentencias SQL que usen la condición de igualdad sobre la clave de partición se ven beneficiadas, el resto escanearan todas las particiones.

- Dependiendo del volumen de sentencias SQL que caigan en esta segundo casuística, escaneo de todas las particiones, el beneficio obtenido por la reducción de la contención de las sentencias de tipo INSERT puede ser menor que trabajo extra asociada el escaneo de todas las particiones.

    - El número de escaneos de particiones de índices es directamente proporcional al número de particiones. Es decir, en vez de tener un índice simple con segmentos no particionados, tenemos N escaneos.

    - Un número alto de particiones puede impactar en el rendimiento de todas las sentencias que no sean del tipo INSERT, por lo que esta técnica puede emplearse exclusivamente después de revisar todos las sentencias SQL que accedan al índice así como la frecuencia de uso de éstos.

- No reduce la frecuencia de la transferencia de bloques al no crear afinidad de éstos a las instancias. Aunque la contención se reduce al incrementar los conjuntos de bloques modificados, éstos se seguirán cambiando en cualquiera de las instancias, por lo que, el rendimiento posiblemente se vea perjudicado ya que las esperas por contenciones locales sobre los bloques siempre son relativamente bajas respecto a esperas por transferencias de éstos.

- Puede incurrir en un mayor uso de la SHARED_POOL, concretamente proporcional al número de particiones.

Particionamiento de tablas por INSTANCE_NUMBER

La idea consiste en:

- Añadir la columna INSTANCE_NUMER a cada tabla y rellenar dicha columna con el número de instancia en curso.

- Esta columna será elegida como clave de particionamiento y se realizarán tantas particiones como instancias de RAC.

- De esta manera, las sentencias de tipo INSERT ejecutadas desde las diferentes instancias se almacenarán en diferentes particiones.

- En caso de que existan índices con contenciones, éstas se reducirán al crearlos como LOCAL al disminuir las transferencias de CURRENT BLOCKS.

Esta técnica es completamente transparente a las aplicaciones y no es necesario cambios a nivel del aplicativo.

Sin embargo, los índices de tipo UNIQUE solo pueden crearse como LOCAL si incluyen la clave de particionamiento o PARTITION KEY. Es decir, si la tabla original tiene un índice UNIQUE sobre la columna DEPTNO, el nuevo índice de tipo LOCAL UNIQUE deberá ser creado sobre las columnas DEPTNO e INSTANCE_NUMBER, cambiando así la restricción de unicidad o de clave primaria, permitiendo duplicar valores de DEPTNO en caso de que sean insertados desde diferentes instancias del RAC.

Ventajas

Entre otras ventajas destacamos las siguientes:

- Reduce enormemente el número de transferencia de CURRENT BLOCKS, reduciendo así las latencias de las sentencias INSERT.

- Reduce la contención por BUFFER BUSY globales para las sentencias de tipo INSERT.

Desventajas

Entre otras desventajas destacamos las siguientes:

- Dado que la clave de particionamiento se realice por un columna que no conoce la aplicación, las sentencia SQL que ésta lance no se verán favorecidas por la eliminación de particiones. En consecuencia, todas las particiones serán escaneadas, por lo que las sentencias SQL y todas las sentencias que no sean del tipo INSERT no escalarán al añadir más instancias al RAC.

- Puede provocar el aumento de transferencia de CR BLOCK cuando la información insertada recientemente es frecuentemente accedida.

- Los cambios a nivel de esquema del modelo de entidad-relación del aplicativo debe mantenerse junto con el versionado del aplicativo.

- En caso de que se añadan instancias al clúster RAC, es necesario realizar cambios a nivel de esquema. Como alternativa, pueden pre crearse tantas particiones como el número máximo de instancias que pensemos tener en el peor de los casos.

- Implica cambios en las restricciones de clave primaria y de unicidad en el caso de que los índices sean de tipo LOCAL.

Tablespaces de solo lectura

Para datos que no se modifican ya que permite minimizar la gestión de bloqueos globales en el clúster al realizar lecturas a disco.

En el caso de existir objetos con datos históricos, es conveniente utilizar el particionamiento de tablas y de sus índices.

En el caso de las particiones con datos históricos, estas pueden ubicarse en TABLESPACES de sólo lectura o READ_ONLY.

Las particiones READ_ONLY, aparte de no requerir de una copia de seguridad diaria, con el consiguiente beneficio a nivel de administración, disminuyen el tráfico entre instancias y permiten un mayor rendimiento de acceso en un entorno RAC.

Índices inversos

En el caso de índices creados contra columnas donde se insertan números de secuencia o valores de fecha, tiempo, o TIMESTAMP, sobre todo en tablas con una elevada actividad de inserción, se puede producir contención en los bloques hoja o LEAF_BLOCKS. Esta contención puede incrementarse en el caso de separación de bloques o BLOCK SPLITS.

Cuando diferentes instancias del clúster están insertando en una misma tabla, las separaciones de bloques hoja o LEAF BLOCK SPLITS pueden provocar importantes contenciones.

Para contrarrestar este efecto, se pueden crear los índices utilizando un índice de Clave Inversa o REVERSE KEY INDEX, especificando la cláusula REVERSE en la sentencia de creación del índice.

Es importante destacar que este tipo de índices no pueden ser usados para hacer un INDEX SCAN, por lo que se recomienda el análisis de su uso antes de establecer la característica de REVERSE.

En el caso de columnas con valores procedentes de de secuencias, éstas pueden aumentar el valor de la cláusula CACHE para que cada instancia genere un rango de valores distinto. Este punto se estudia en un apartado específico de este mismo documento.

Dependiendo de la frecuencia de acceso y el número de procesos que acceden concurrentemente para la inserción de datos, los índices pueden convertirse en puntos calientes y la contención puede venir provocada por:

- Índices con valores monótonamente crecientes, que producen un crecimiento desbalanceado del éstos o RIGHT GROWING TREES.
- Frecuentes LEAF BLOCK SPLITS, convirtiéndose en puntos de serializarían o de botella en el caso de constantes LEAF BLOCK SPLITS de un mismo árbol.
- Índices con árboles con poca profundidad, donde la mayoría de los bloques son accedidos desde el nodo raíz.

Por tanto como recomendación general, para aliviar el impacto en el rendimiento de la generación de bloques calientes y/o LEAF BLOCK SPLITS en el acceso a los índices es la homogenización de la estructura de éstos. Para ello, podemos establecer las siguientes recomendaciones:

- Particionamiento por GLOBAL HASH de los índices.
- Incrementar la cache de las secuencias implicadas.
- Crear índices de clave inversa.

```
ALTER INDEX <index_name> REBUILD REVERSE;
```

Ventajas

Entre las ventajas del uso de índices inversos podemos destacar:

- Relativamente fáciles de construir ya que exclusivamente los segmentos del índices son modificados en el proceso de reconstrucción.
- Reducen la contención por BUFFER BUSY al dispersas las sentencias de tipo INSERT a los largo de las diferentes hojas del árbol de los índices.
- Reducen las contenciones por BUFFER BUSY y CACHE BUFFER CHAINS de forma general.

Desventajas

Entre las desventajas del uso de índices inversos podemos destacar:

- Dado que valores similares pueden almacenarse en bloques diferentes de un mismo índice, éste solo puede usarse con cláusulas WHERE que contengan el predicado de igualdad sobre la clave completa del índice. Es decir, por ejemplo, si sobre la columna COL se construye un índice inverso, sentencias con clausulas WHERE del tipo WHERE COL=VALUE podrán usar dicho índice, pero otras del tipo WHERE COL>VALUE no podrán.

- No reducen la frecuencia de las transferencias de bloques, ya que no crean afinidad entre éstos y las instancias.

Uso de secuencias.

Los índices generados con valores procedentes de secuencias suelen tender a sufrir contenciones de tipo LEAF BLOCK cuando el índice de inserciones sobre ellos es alta. Esto se debe a que la hoja del árbol que contiene el máximo valor del índice cambia con cada inserción realizada ya que el valor crece monótonamente siguiendo la secuencia. En entornos basados en RAC, esto implica que el numero de CURRENT y CR BLOCKS que se transfiera entre los nodos sea alto.

Para evitar esta situación, la técnica más fácil es habilitar e incrementar el cacheo de las secuencias. Si el valor de este cacheo es alto, puede provocar la afinidad de los bloques del índice en las secuencias divisiones de las hojas de éste.

- Si el valor del cacheo de la secuencia se establece al valor por defecto, 20, mientras que una instancia inserta los valores 1, 2 y 3, la segunda instancia estará insertando concurrentemente los valores 21, 22 y 23. Dado que la diferencia entre los valores no es grande y dado que dentro un bloque pueden existir multitud de valores, al final, las instancias terminarán intercambiándose estos bloques.

- Si por el contrario establecemos el valor de la cache a 50.000, mientras la primera instancia inserta los valores, 1,2 y 3, la segunda insertará los valores, 50.001, 50.002 y 50.003. Al principio, las dos instancias escribirán en el mismo bloque, pero después de varios divisiones de las hojas del índice, la tendencia cambiará y diferenciará claramente los bloques de índices usados por cada instancia.

No es fácil establecer un valor recomendado para el valor de cacheo de una sentencia, por lo que normalmente éste se calcula en función del ratio de inserción que sufra el índice, cuando más alto, mayor será la cantidad de cacheo.

Ventajas

- De fácil configuración y sin impacto en la disponibilidad ya que no es necesario reconstruir ningún segmento.

- Reduce el tráfico de la global cache así como la contención por global buffer busy al crear afinidad entre el bloque y la instancia.

Desventajas

- Solo aplicable a índice derivadas de valores procedentes de secuencias.

- Solo aplicable en el caso de que el orden no sea fundamental.

- Pueden crear grandes huecos dentro de los valores de un secuencia, provocado al detener las instancias y volver a iniciarlas.

Las secuencias son usadas frecuentemente por el aplicativo para crear una secuencia numérica. Las secuencias tiene dos propiedades que afectan directamente al rendimiento de éstas: el cacheo y la ordenación. Cada una de estas propiedades básicamente se habilitan o se deshabilitan, y por defecto las secuencias no se ordenan y tiene una cache con un tamaño de 20. Como recomendación general, las secuencias deberían usarse con CACHE y con un tamaño razonablemente grande, tanto con la opción NOORDER como con la opción ORDER.

Históricamente, la documentación sobre secuencias ha mantenido que las secuencias con las opciones CACHE y ORDER no están soportadas en configuraciones en RAC, con afirmaciones como que el cacheo es deshabilitado internamente cuando se usan secuencias con la cláusula ORDER. Esta y otras afirmaciones similares son totalmente falsas.

A continuación veremos las siguientes opciones disponibles en cuanto al uso de secuencias en entorno RAC.

CACHE+NOORDER.

Esta es la configuración con menor impacto de rendimiento en configuraciones en RAC, y es la configuración por defecto cuando no se especifican otras opciones. En este caso, cada instancia cachea un conjunto de números distinto. Dado que no se ordenan, se pueden producir saltos en la secuencia en caso de que las instancias se detengan de forma no controlada.

CACHE+ORDER

En este caso cada instancia cachea el mismo conjunto de números, por tanto la ordenación de éstos está garantizada pero pueden producirse saltos en la secuencia. Esta configuración se comporta desde el punto de vista de rendimiento, mejor que cualquier opción con NOCACHE. La sincronización/comunicación del estado del cacheo de la secuencia se realiza mediante un bloqueo de instancia de tipo SV sobre el recurso en cuestión y es gestionado por el proceso interno LCK. Éste coordina el bloqueo que cede al proceso foreground de sesión que lo ha solicitado para generar un nuevo valor de la secuencia.

Para secuencias que se usen frecuentemente, el tamaño de sus caches debería incrementarse directamente proporcional a su uso. En caso contrario, es decir, un no cacheo o un cache con un tamaño bajo para su uso, provocará contenciones de tipo SQ que podrán observarse en la vista del diccionario V$ENQUEUE_STATISTICS.

Un claro ejemplo de esta situación es la secuencia AUDSES$. Ante masivas conexiones a la base de datos se pueden provocar las contenciones comentadas anteriormente si se deja con su valor por defecto. El valor por defecto de esta secuencia se ha incrementado hasta 10.000 en Oracle RDBMS Server 10gR2 y superiores.

NOCACHE+NOORDER.

El no-uso de este tipo de configuración suele está ligado a alguna justificación o limitación de la lógica de negocio del aplicativo, que hacen necesario la generación de secuencias sin saltos. Sin embargo, la ordenación no está garantizada. El

rendimiento de esta configuración frente a la configuración NOCACHE+ORDER es superior.

NOCACHE+ORDER

Esta configuración es usada cuando es estrictamente necesario la generación de una secuencia numérica ordenada y sin saltos. Tanto la ordenación como la continuidad de la secuencia está garantizada. Esta configuración es la que peor rendimiento posee en configuraciones en RAC.

Otras consideraciones

Los efectos colaterales de la activación de la ordenación y/o cacheo de secuencias en RAC puede detectarse como altos tiempos de espera de tipo ROW CACHE LOCK sobre DC_SEQUENCES. Normalmente el número de solicitudes GET, modificaciones, peticiones GES y conflictos GES suelen coincidir con el número de ejecuciones de sentencias SQL que generan nuevos números de una secuencia.

Los índices con valores clave generados por secuencias tienden a tener problemas de contenciones en los bloques hojas cuando la frecuencia de inserción es alta.

Esto se debe a que los bloques de las hojas de éstos almacenan el valor más alto y éste crece monótonamente en cada inserción de datos. En el caso de configuraciones en RAC, esto puede llevar a una alta transferencia de bloques current y CR entre los nodos y por tanto a un bajo rendimiento del sistema en general.

Esta situación suele tener los siguientes síntomas:

- Peticiones CR y esperas de tipo BUFFER WAITS sobre cabeceras de segmentos de UNDO no locales.
- Esperas por enqueues de tipo TX y esperas de tipo BUFFER BUSY WAITS sobre ramas u hojas de índices.

SQL Execution

En la práctica todas las directivas y best practices de ajuste de rendimiento de SQL que aplican en bases de datos mono instancias, aplican de igual manera en entornos basados en RAC, con la salvedad que en el segundo caso, algunas de ellas, se hacen indispensables. Veamos algunas de ellas:

Fallos de parseo

Se ha comprobado, que los fallos de parseo puede limitar la escalabilidad de un sistema hasta en un 30%. Los fallos de parseo provocan que si la sentencia no es 100% correcta, por ejemplo, su ejecución retorna un ORA-904 por ambigüedad en una columna, o bien no tiene los permisos necesarios para acceder a una tabla.

Puede comprobarse en qué grado un sistema está viendo afectado por este problema a través de la sentencia SQL siguiente, donde idealmente su valor debería ser 0.

```
SELECT name, value FROM v$sysstat WHERE name = 'parse
count (failures)';
```

En ocasiones es difícil localizar las partes del aplicativo que causan estás situaciones, pero existen dos formas de aislar el problema.

La primera de ellas es consultar las estadísticas de la misma métrica pero a nivel de sesión, por ejemplo, usando la siguiente SQL.

```
SELECT st.sid, sn.name, st.value FROM v$sesstat st,
v$statname sn WHERE st.statistic# = sn.statistic# AND
sn.name = 'parse count (failures)' AND st.value > 0 ORDER
BY 3 ;
```

La segunda es tracear la sesión estableciendo evento a nivel de sesión que permite que se genere un traza con información extra por cada error de tipo ORA-904 error, como se muestra a continuación:

```
oradebug setospid <OSPID >

oradebug event 904 trace name errorstack level 1

oradebug tracefile_name
```

Del análisis de este fichero de trazas resultantes de la activación del evento se podrá obtener las sentencias que generan los errores de parseo.

Auditoría

En entornos RAC, habilitar la auditoría puede conllevar efectos colaterales negativos ya que dicho proceso provoca gran cantidad de SHARED LIBRARY CACHE LOCKS, que depende del uso de que se haga del sistema de auditoría y la cantidad de funcionalidades que se auditen, pero puede llegar a un aumento de hasta el 50%.

Por tanto, no se recomienda establecer auditoria en entornos basados en RAC si no es estrictamente indispensable. En dicho caso, se recomienda minimizar los objetos y/o operaciones a auditar.

Disparadores o triggers de tipo Logon/Logoff

A partir de Oracle 8i, los disparadores o TRIGGERS de tipo LOGON y LOGOFF puede definirse a nivel de base de datos. El uso de este tipo de disparadores y el uso de la auditoría a la misma vez, puede tener efectos negativo en el rendimiento de la base de datos y en particular en el número de LIBRARY CACHE LOCK que deben realizarse.

Este último punto es de vital importancia en entornos basados en RAC, por lo que en la medida de lo posible deben evitarse.

Operaciones Full Table Scan

Cualquier operación de tipo FULL SCAN siempre es costosa y más en entornos basados en RAC donde hay que sumar el tráfico a través de la interconexión entre instancias.

Por tanto, sin que ello conlleve la erradicación de este tipo de operaciones, si es importante minimizarlas, ya que de lo contrario, en entornos RAC, el servicio de GLOBAL CACHE tendrá que atender a numerosas peticiones de bloques incrementando significativamente e innecesariamente el tráfico del interconnect.

Para comprobar el número de sentencias que usan FULL SCAN sobre tablas de gran tamaño, podemos usar la siguiente SQL:

```
SELECT name, value FROM v$sysstat WHERE name = 'table
scans (long tables)';
```

En el caso de que la SQL anterior indique que se están realizan FULL SCAN, el siguiente paso es comprobar si los planes de ejecución son o no razonables. Podemos usar la siguiente SQL:

```
SELECT object_owner, object_name, bytes FROM v$sql_plan
WHERE operation = 'TABLE ACCESS' AND options = 'FULL'
AND object_owner NOT IN ('SYS', 'SYSTEM');
```

Otra forma de identificar las sentencias SQL que son costosas desde el punto de vista de consumo de CPU y memoria es usar la vista V$SESSION_LONGOPS. A continuación se muestra un ejemplo:

```
SELECT sl.sid, sl.serial#, sl.opname,sl.sofar,
sl.totalwork, sl.time_remaining, sl.username,
sl.sql_hash_value FROM v$session_longops sl WHERE
sl.totalwork > 50000 AND sl.time_remaining > 0 ORDER BY
sl.totalwork desc;
```

Esta vista almacena información sobre escaneo de tablas, hash joins, operaciones relacionadas con Advanced Queueing y operaciones del RECOVERY MANAGER. La ventaja de esta vista radica en que su información no es eliminada de forma inmediata tras la finalización de la operación.

Objetos PL/SQL

La compilación nativa de código PL/SQL fue introducida como funcionalidad en Oracle 9i y permite que estos objetos no se almacenen en el diccionario y sean compilados como librerías compartidas y localizadas a nivel de FILE SYSTEM.

A partir de Oracle 10g, las copias compiladas de las librerías compartidas pueden almacenarse en la propia base de datos salvando así la limitación de la versión anterior que necesitan un FILE SYSTEM compartido. Una vez almacenadas en la base de datos, éstas son propagadas al FILE SYSTEM o en clúster de todos los nodos

Paralelismo y SQL

La ejecución paralela o Parallel Execution (PX) permite realizar ciertas operaciones en paralelo, entre otras consultas SQL, CREATE TABLE AS SELECT, INSERT AS SELECT, creación de índices, recovery paralelo, DML paralelo sobre tablas particionadas y creación de estadísticas en parelelo.

Este tipo de operaciones suele tener un comportamiento estable desde el punto de vista de escalabilidad en sistemas SMP, y en el caso de RAC extienden este comportamiento entre las diferentes instancias del clúster.

Cambio en PX a partir de Oracle 10g

En PX el modelo empleado para las sentencias SQL ha cambiado desde el modelo donde existía una serie de esclavos SQL coordinador por un proceso padre a un modelo basado en un único cursor tratado de forma parelela, Parallel Single Cursor (PSC).

En esta caso no es necesario el coordinador (QC) para construir la sentencia SQL para cada uno de los DFO o esclavos en cada una de las instancias, donde cada esclavo necesita tener, parsear y ejecutar su propio cursor.

En el nuevo modelo basado en PSC, se construye y compila un único cursor por instancia que contiene todo lo necesario para permitir la ejecución paralela de éste.

De esta forma se permite que cada esclavo dentro de la misma instancia comparta el mismo cursor con la consecuente mejora en el rendimiento y en el consumo de memoria.

Grados de paralelismo o DOP

Los diferentes grados de paralelismo o Degree Of Parallelism (DOP) que usa una sentencia determinada se basa en diferentes criterios como el tipo de sentencia, SQL, DML, DDL, los objetos implicadoss, tablas o índices, etc.

En el caso de sentencias SQL, es usual ver el DOP como el máximo de las tablas implicadas, es decir, si la tabla A tiene un grado de 4 y la tabla B tiene uno de 6, una sentencia SQL que realice un JOIN entre ellas, usará un DOP de 6 para escanear ambas.

El número de proceso encargados de la ejecución paralela para una determinada sentencia con un determinado DOP dependerá igualmente del tipo de operación, ya que el modelo de ejecución permite establecer pipeline entre dos procesos,

permitiendo así enrutar la salida de un proceso hacia otro proceso, por ejemplo, en ordenación paralelas o entre operaciones de tipo JOIN.

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.

Tuning automático de PX

A partir de Oracle 10g, el parámetro PARALLEL_AUTOMATIC_TUNING se considera en estado “deprecated”, y por tanto, no se recomienda establecer explícitamente este parámetro. Oracle realizará la elección de los parámetros por defecto más razonable en cada de las situaciones, permitiendo así el uso de PX sin la necesidad de parametrización y ajuste previos. En Oracle 12cR2 el parámetro ha sido eliminado y des-soportado a partir de esta versión.

Por defecto, Oracle configura suficientes procesos para ejecución paralela, habilita la posibilidad de compartirlos entre diferentes usuarios y ajusta el DOP en función de la carga de usuarios.

En el caso de entornos RAC, los procesos de PX locales tiene prioridad para realizar operaciones paralelas frente a los remotos con el fin de reducir la carga de comunicación entre las instancias. Sin embargo, si los recursos locales no son suficientes, se usarán los procesos remotos necesarios.

En este caso, se trazará un advertencia en el fichero de alertas de la base de datos, alert.log, indicando que la ejecución paralela solicita mas procesos de los establecidos a través del parámetro PROCESSES del fichero de parámetros de la base de datos, tal como se muestra a continuación:

```
Adjusting the {default|requested} value of parameter
parallel_max_servers from %d to %d due to the value of
parameter processes (%d)
```

Procesos para ejecución paralela

El algoritmo que se sigue para establecer donde crear los procesos para ejecución paralela sigue las siguientes primitivas:

- En tiempo de arranque de la instancia, ésta publica a todo el clúster de cuantas CPU dispone y el valor del parámetro PARALLEL_MAX_SERVERS establecido en su init.ora.
- Cada instancia periódicamente publica a todo el clúster su carga de trabajo, en principio cada 300 milisegundos y solo si existen cambios significativos. Esta información incluye información sobre los procesos que atienden peticiones de clientes o foreground process, mas procesos paralelos y procesos coordinadores.
- Cuando se inicia una operación en paralelo, por ejemplo, con DOP de 20, la instancia donde se inició esta operación, recoge la información de carga de todas las instancias disponibles y las ordena en base a ella.
- Las solicitudes de procesos de tipo PX se realizan en grupo de estos desde la instancia con menos carga hasta la más cargada hasta cumplir la solicitud. En este proceso se tiene en cuenta la asignación usando la menor cantidad de instancias posibles con el fin de minimizar la comunicación inter-instancia.
- Al mismo tiempo, se intenta evitar cualquier situación de desbalanceo de carga, es decir, para una solicitud con DOP de 24 procesos de tipo PX en un clúster de 2 instancias, siempre intentará repartir la solicitud de 12 procesos por nodo, frente a una reparto, por ejemplo, de 16 y 8.

Ejecución paralela entre instancias.

Parallel Query

Por defecto, todas las instancias de una base de datos en RAC son candidatas para realizar operación de ejecución paralelas.

En esta situación, hay que tener sumo cuidado a la hora de establecer las condiciones de ejecución paralela, ya que pueden darse situaciones donde la totalidad de las instancias se queden sin procesos PX disponibles y las peticiones lleguen a ejecutarse con un grado muy inferior de paralelismo e incluso de modo secuencial.

Veamos un ejemplo de esta situación:

- Clúster de 2 nodos
- PARALLEL_MAX_SERVERS = 4

- Las dos instancias ejecutan simultáneamente:

```
select /*+ PARALLEL(mytable,8) */ count(*) from mytable;
```

En esta situación, la instancia A creará 4 procesos paralelos en la instancia A y otros 4 en la instancia B. Dado que el parámetro PARALLEL_MAX_SERVERS es igual a 4, la instancia B se queda sin la posibilidad de crear los proceso PX necesarios, y por tanto ejecutará la sentencia de forma secuencial, provocando así que la misma sentencia se ejecute más rápido en la instancia A que en la B.

Es importante destacar que cualquier ejecución paralela provoca que los datos se lean directamente desde disco, DIRECT PATH READ, hasta la PGA, dejando a una lado la BUFFER CACHE.

Este tipo de operaciones de tipo DIRECT PATH READ se desactivan si la tabla en cuestión es pequeña respecto a la BUFFER CACHE total. El tamaño de la BUFFER CACHE y el parámetro _SMALL_TABLE_THRESHOLD controlan este ratio. Finalmente, esta misma política aplica en otras operaciones de tipo UPDATE/INSERT donde se debe actualizar los valores en la cache.

Parallel DML

Las operaciones DML paralelas trabajan de la misma manera. En el ejemplo siguiente, 12 procesos de tipo PX se crearán en las 3 instancias disponibles para realizar la operación de UPDATE:

```
SQL> alter session enable parallel dml;

Session altered.

SQL> alter session set "_px_trace"="all";

Session altered.

SQL> update /*+ parallel(ware,12) */ ware set w_ytd=w_ytd;
```

Uso de la funcionalidad de Parallel Instance Groups

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.

Otras consideraciones en entornos RAC

Afinidad de procesos

Una de las técnicas más usadas en entornos de clústeres basados en RAC y especialmente en su predecesor, Oracle Parallel Server (OPS), es crear una afinidad de los procesos a las diferentes instancias del clúster.

Con esta afinidad se busca el particionamiento lógico de los procesos de negocio o de las aplicaciones transaccionales conectadas al clúster. Este particionamiento reduce la contención por la compartición de recursos, la disminución de la comunicación entre las instancias, y por tanto aumenta el rendimiento de los aplicativos.

Si bien este particionamiento no siempre es posible, tal como se ha comentado anteriormente, en caso de serlo, puede conseguir grandes resultados desde el punto de vista de concurrencia y tiempo de respuesta global de las aplicaciones, simplemente realizando un estudio previo del tipo de carga y uso de tiene la base de datos.

Por ejemplo, crear afinidad de procesos de carga de datos, modificaciones masivas de datos, cálculo de estadísticas, y en general en todas aquellas situaciones propensas a la contención, ya sea por puntos de serialización o por consumo de grandes cantidades de datos.

Esta afinidad es necesaria, o al menos conveniente, en otras situaciones que no están directamente vinculadas a las aplicaciones, pero que poseen una importancia clave en la administración y uso de la base de datos.

A continuación se muestran algunos de estos casos:

Dbms_scheduler frente Dbms_job

A partir de Oracle 10gR1, aparece Unified Scheduler (ahora llamado Oracle Scheduler), implementado en el paquete DBMS_SCHEDULER como sustituto de DBMS_JOB para la gestión y ejecución de tareas programadas de replicación, AQ y Streams.

Ahora en 12cR2 el paquete DBMS_JOB ha sido deprecado y será des-soportado en una release futura; por lo que hay que cambiarse a DBMS_SCHEDULER.

Algunas de las ventajas de DBMS_SCHEDULER frente a DBMS_JOB en entornos RAC son las siguientes:

- Los trabajos o tareas son asociadas a un servicio, si la instancia asociada con el servicio fall, el trabajo acompaña al servicio hasta su instancia de backup.
- Cada instancia de RAC tiene su propio coordinador de trabajos que ejecutan los asociados al servicio que soportan.

- Las estadísticas de los servicios combinadas con funcionalidades de DBMS_SCHEDULER permite la gestión eficiente de los trabajos de replicación en entornos RAC.

Dbms_job

DBMS_JOB es un paquete que suele usarse con bastante frecuencia en entornos donde existe replicación de datos con algunas de las opciones incluidas en Oracle Advanced Replication.

Si las tablas replicadas son exclusivamente accedidas desde uno de las instancias, tiene sentido usar la funcionalidad de afinidad a una instancia implementada en el paquete DBMS_JOB. Esto se consigue al especificar el número de la instancia a las operaciones de gestión de los trabajos. Los procedimientos que hacen uso del parámetro INSTANCE son los siguientes:

- DBMS_JOB.SUBMIT
- DBMS_JOB.CHANGE
- DBMS_JOB.INSTANCE
- DBMS_JOB.USER_EXPORT

Adicionalmente, la asignación de una instancia de un trabajo, puede cambiarse en cualquier momento a través del procedimiento DBMS_JOB.INSTANCE.

Estas recomendaciones no solo aplican en entornos de replicación, ya que por regla general cualquier tipo de trabajo se ve beneficiado al trabajar en un única instancia.

Advanced Queuing

Como norma general, se puede recomendar que en el caso de que los proceso de encolado y desencolado se realicen sobre una misma instancia, en tiempo de creación de las colas implicadas, se indique esa misma instancia.

Hasta la versión 10g, el proceso que espera para descolar un mensaje, no es avisado de su llegada en caso de que éste se encole en una instancia diferente. A partir de 10g, esta restricción ha sido eliminada gracias a la aparición de un nuevo proceso encargado de la comunicación entre las instancias y responsable del "global posting" de los mensajes.

A pesar de que esta funcionalidad incrementa la eficiencia de los procesos que desencolan mensajes desde una instancia diferente donde fueron encolados, sigue existiendo una afinidad de la cola contra una instancia, por lo que los procesos de encolado, INSERT, y los procesos de desencolado, DELETE, seguirán necesitando transferencias de current block y eventos de global cache por BUFFER BUSY WAITS.

Existen determinadas situaciones donde las colas son más susceptibles de particionarse. Para que esto sea posible deben cumplir los siguientes requisitos:

- Todos los clientes que encolen y desencolen en una cola o en una de sus particiones deberían ser capaces de concentrar todos sus accesos a través de una única instancia de clúster.

- En el caso de los accesos se realicen a través de una condición, por ejemplo ID de mensajes, pueden que algunos mensajes no estén en dicha instancia. Por tanto, no puede existir criterio de selección en el proceso de desencolado.

- En una cola particionada, las diferentes partes de ésta son accedidas de forma independiente. Sin embargo, existe un latencia el procesado de los mensajes que provoca que lo mensajes encolados en la cola lógica no sean desencolados respetando el orden de encolamiento.

- En el caso de que sea necesaria una serialización extricta, el grupo de mensajes que necesiten esta característica deberán encolarse en la misma instancia. En esta situación, el grupo de mensajes será desencolado respetando el orden.

Tecnologías no soportadas o con limitaciones

Oracle Streams

Los procesos de Streams se ejecutan en instancia que tiene asignada como OWNER. Esta característica puede consultarse a través de la vista DBA_QUEUE_TABLES y su columna OWNER_INSTANCE. En caso de que la instancia asignada falle, esta propiedad es automáticamente traspasada a una de las instancias supervivientes.

En el caso de la propagación de eventos de Streams en entornos RAC, los database links deben estar configurados en la instancia particular que posee la cola. Si la instancia falla, al contrario que la propiedad, el database link necesitará recrearse de forma manual, es decir, drop database link y create database link con la información de conexión correcta.

En este escenario, la propagación desde la base de datos origen no será capaz de desencolar mensajes y enviarlos a la de destino hasta que el database link funcione correctamente hasta la instancia con la propiedad de la cola.

Paquete PL/SQL Dbms_pipe

El paquete PL/SQL DBMS_PIPE se basa en las capacidades de los sistemas operativos subyacentes en crear tuberías o pipe de comunicación entre dos procesos.

Este mecanismo permite no solo la comunicación, sino que ésta pueda tener un carácter asíncrono y desacoplado siguiendo el paradigma de productor-consumidor.

Teniendo en cuenta esto, y que la arquitectura de referencia para bases de datos en clúster establece el uso de las funcionalidades de balanceo de carga para las sesiones, no se permitirá el uso de dicho paquete PL/SQL, ya que no sería posible asegurar que las dos sesiones de bases de datos conectadas a la pipe pertenezcan a la misma instancia.

Tablas externas.

Se entiende como tablas externas a conjuntos de datos que se encuentran almacenados en ficheros de texto plano que a su vez residen en un sistema de ficheros accesible desde la instancia de base de datos.

En este caso, ante la necesidad de usar este tipo de almacenamiento, es necesario asegurar que todas las instancias que componen el clúster tienen acceso al sistema de ficheros donde se almacenan este tipo de tablas.

Si en un principio esto es posible, el uso de tablas externas, ya sea en entornos de clústeres como fueras de ellos, queda excluido de la arquitectura y del catálogo de normas de uso de bases de datos.

Análisis de esperas por enqueues

Las esperas por enqueues pueden convertirse en esperas globales debido al propio funcionamiento de RAC y hay que tener en cuenta que ciertos enqueues son más visibles en este tipo de configuraciones que en configuraciones en single-instance.

En el caso de los eventos de espera sean frecuentes y de alto impacto en el rendimiento del sistema, la recomendación es contactar con los servicios de soporte de Oracle.

A través del análisis de la información de depuración que los eventos 10704 y 10706 generarán, se podrá obtener una imagen más clara de las causas y por tanto de las soluciones para los problemas de rendimiento.

Se recomienda el uso de ambos eventos de espera de forma conjunta, ya que el evento 10706 supervisa la actividad de los global enqueue y el 10704 supervisa exclusivamente las operaciones de enqueue locales.

Con carácter general no se recomienda la activación de estos eventos a menos que sea bajo la recomendación y supervisión de Oracle Soporte.

Principales enqueues en entornos RAC

Sin embargo existen una serie de eventos de espera, que por su naturaleza o por la configuración en RAC, son típicos y tanto su análisis como las recomendaciones para evitarlos o eliminarlos se tratan a continuación.

SQ

Se tratará en un punto de este documento debido a su extensión.

TA

Este enqueue está típicamente asociado a configuraciones en RAC y permite la recuperación de transacciones como parte de la recuperación de una instancia dentro del clúster.

En el caso de que este enqueue TA aparezca, normalmente en versiones 10g o superiores, puede ser un síntoma claro de que el SMON está realizando la recuperación de transacciones. Si estas esperas aparecen frecuentemente o son de larga duración, se debería estudiar las causas junto con los servicios de soporte de Oracle.

US

El enqueue de tipo US es principalmente usado por la opción Automactic Undo Management (AUM) y son usados cada vez que creamos, habilitamos, deshabilitamos o reducimos un segmento de rollback.

En un entorno con una alta carga de transacciones concurrentes y/o con un largo tiempo de ejecución de estas transacciones, el número de segmentos de rollback crece y decrece rápidamente y dado que las operaciones sobre los segmentos de rollback no pueden ser controladas cuando se usa AUM, este tipo de enqueue puede convertirse en una porción importante en el global de esperas del sistema.

El principal proceso que solicita enqueue de tipo US es el proceso interno SMON. Este proceso solicita este enqueue para comprobar si el segmento de rollback puede o no habilitar o deshabilitarse (online y offline). Recordar que esta comprobación se realiza independientemente si está o no configurado AUM.

Otras esperas por enqueues en RAC

En este apartado se centra en el análisis de las causas de otras esperas de tipo enqueues.

TX

Existe cierto patrón de eventos de espera y estadísticas que son características de la serialización debido a la fragmentación de índices. Esta situación puede provocarse por la carga de trabajo y por la alta cantidad de datos insertados. Esta situación puede ser dramática en el caso de que el número de procesos que acceden a los índices de forma concurrente sea alta y puede convertir a los índices en verdaderos puntos calientes y por tanto, crear un problema de rendimiento.

Esta situación puede provocarse, entre otras causas, por:

- Índices que se incrementa de forma monótona según una clave, como por ejemplo, una secuencia
- Fragmentación de los bloques que contienen las hojas de los árboles B-Tree sobre los que se soportan los índices.
- Escasa profundidad del árbol del índice, que provoca que todos los accesos al árbol se realicen a través de la hoja raíz.

Estas situaciones puntos globales calientes sobre bloques de índices y la fragmentación de las hojas de los índices pueden evitarse siguiendo las siguientes recomendaciones:

- Uso de particiones por HASH o por rango.
- Si las claves son generadas a partir de una secuencia, incrementar la cache de la secuencia. Este punto se tratará con detalle en un apartado de este mismo documento.
- Añadir el número de la instancia de la base de datos a las claves.
- Uso de distribuciones uniformes o aleatorias para creación de las claves.

HW

En ciertas ocasiones se presentan eventos de espera y estadísticas inherentes a la actividad de la aplicación, en especial cuando ésta tiene como fundamental función la inserción de datos que crean nuevos bloques que a su vez son alojados en un segmento.

En Oracle, la marca de agua o HIGH WATERMARK (HWM) de un segmento es el puntero a un bloque de datos donde existen bloques formateados y libres para la inserción de un dato nuevo. Si el aplicativo inserta datos de forma intensiva, se debe crear nuevos bloques después de buscar en las freelists o en los bloques de tipo L1 y no encontrar espacio libre.

Este proceso conlleva el formateo de los bloqueos, insertarlos en la cabecera del segmento y avanzar la marca de agua. Estas tareas se realizan mientras se mantiene un enqueue de tipo HWM de forma exclusiva en los nuevos bloques que se libera al completarse la operación.

Los síntomas más comunes de esta situación son:

- Un alto porcentaje esperas por el enqueue HWM
- Un alto porcentaje esperas por eventos GC CURRENT GRANT.

Estos síntomas están provocados por la serialización del uso de la HWM con la consecuente ralentización de todo el proceso de creación de nuevos bloques. En un entorno basado en RAC, la duración de esta operación de gestión de espacio es proporcional al tiempo necesario para solicitar y adquirir los enqueue de tipo HWM más el tiempo necesario para adquirir bloqueos globales para todos los nuevos bloques. Este tiempo es normalmente corto, ya que en circunstancias normales no existen conflictos entre los nodos por los bloques nuevos.

Para encontrar la causa de este tipo de problemas, y determinar qué segmento es que esta creciendo más rápido, podemos optar bien por extrapolar las ratios de datos insertados a través de la vista V$SQLAREA o bien capturando las columnas ID1 y ID2 de los enqueue de tipo HWM, donde ID1 es el número del tablespace donde reside el bloque e ID2 es la dirección del bloque de la cabecera del segmento.

El estudio de las vistas V$ENQUEUE_STAT, V$SQLAREA y V$SESSION_WAIT así como los eventos 10046 y 10706 permiten la localización del segmento en cuestión.

Como se mencionó anteriormente, escenario donde este tipo de eventos de espera se producen es en aquellos donde la carga de datos sea una de las tareas fundamentales de la lógica de negocio, y por tanto la gestión de espacio debe acelerar en la medida de lo posible.

Por tanto, se realizan las siguientes recomendaciones en este sentido:

- Se recomienda definir un tamaño de extensión grande e uniforme para los segmentos manejados localmente y con gestión automática de espacio si estos son candidatos a grandes inserciones de datos.
- Si la concurrencia de procesos de instancias no locales es baja, se recomienda usar un tamaño de bloque grande para reducir la cantidad de creación de bloques nuevos.
- Si aplica, use grupos de FREELISTS para los bloques de datos.
- Es este caso, y si la concurrencia entre instancias es baja, usar el pre alojo de extensiones y bloqueos usando funciones HASH para alinear los bloqueos con los límites de las extensiones.
- Tan solo recordar, que en caso de usar grupos de FREELISTS, puede provocarse la fragmentación de bloques en caso de que se eliminen filas o se añadan o eliminen instancias a la base de datos y por tanto, los bloques de datos con afinidad a un nodo en especial pueden terminar en la FREELIST de otra instancia.

- Por tanto, de forma genérica se recomienda la gestión de espacio automática o AUTOMATIC SEGMENT SPACE MANAGEMENT (ASSM) en la creación de tablespaces y en especial en bases de datos en RAC.

Eventos de espera de tipo Global Cache

En este apartado del documento se realiza un breve repaso de los eventos de espera de tipo GLOBAL CACHE.

gc current/cr request,

este tipo de eventos de espera son solo relevantes mientras se realiza un petición gc request sobre un cr o sobre current buffer, ya que actúan como marcadores hasta que la petición se completa.

gc [current/cr] [2/3]-way,

se produce cuando un current o un bloque cr se ha solicitado y recibido después de 2 o 3 saltos de red, en esta caso sin ningún tipo de espera o congestión.

gc [current/cr] block busy,

se produce cuando un current o un bloque cr se ha solicitado y recibido, pero no fue enviado por los procesos LMS de forma inmediata por alguna condición que provoco la dilación en el envio..

gc [current/cr] grant 2-way,

se solicitó un current o bloque cr y se recibió el mensaje de concesión del permiso, y éste fue otorgado sin ningún tipo de retraso. En el caso de que el bloque no esté en la cache local, este tipo de concesión es seguida de una lectura a disco desde la instancia solicitante.

gc current grant busy,

se solicitó un current o bloque cr y se recibió el mensaje de concesión del permiso. Sin embargo, la concesión se realizó tras una espera ante un bloqueo y no pudo atenderse de forma inmediata.

gc [current/cr] [block/grant] congested,

se solicitó un current o bloque cr y se recibió el mensaje de concesión del permiso. Sin embargo, existió algún tipo de congestión que implicó que la petición empleara mas de 1 ms durante su proceso interno.

gc [current/cr] [failure/retry],

tras la solicitud de un bloque, se recibe un fallo como respuesta u ocurre cualquier otras situación excepcional.

gc buffer busy,

si el tiempo entre accesos a un buffer llega a ser menor que el tiempo que éste permanece en estado pinned, se dice que el buffer que contiene al bloque está ocupado y como consecuencia las sesiones de usuarios deben esperar a que pase a estado unpinned.

gc cr grant 2-way, db file sequential/scattered reads.

En situación poco frecuente donde la distribución de la aplicación entre las instancias del clúster no facilita la compartición de los datos o ésta se realiza infrecuentemente, es difícil que se reciban bloques de otras instancias a través del InterConnect y la gran mayoría de las lecturas se realizan a disco.

Los síntomas de este tipo de escenarios son los siguientes:

- Gran número de lecturas físicas.
- Presencia de escaneos completos de tablas (FULL TABLE SCANS y LONG FULL TABLE SCAN).
- Eventos de tipo GC CR WAIT que consume la mayor parte del tiempo total de espera.
- Eventos db_file_sequential/scattered_read que consumen un porcentaje considerable del tiempo total de espera.

Además, en estas situaciones suele darse un mayor consumo de CPU y por tanto, de los tiempos de respuesta del aplicativo, debido a las peticiones remotas de tipo CR entre los nodos del clúster. Para determinar si esta casuística es importante o no es un sistema, generalmente se suele usar la herramienta TKPROF para formatear la salida del evento 10046 y así determinar si la sentencia SQL está o no siendo afectada por la I/O a disco y por las peticiones CR.

Para este tipo de escenarios, se recomienda:

- Realizar un ajuste de las sentencias SQL para mejorar el porcentaje de acierto local y reducir la I/O de disco.
- Separar los segmentos de datos que son más frecuentemente accedidos en diferentes tablespaces y/o particiones y crear tablespaces con tamaños de bloque mayores.

Esta sobrecarga de trabajo debido a las peticiones globales CR son debido a que las sentencias SQL fallan en las caches locales. Este fallo lleva consigo un intento de localizar el dato en la cache del resto de las instancias, ya que la probabilidad de que este dato esté en otra caché es alta. Esta búsqueda implica un mensaje global, pero en muchos casos, también conlleva un bloqueo de tipo S y una lectura a disco, más el consumo de CPU y la latencia de la petición global. Dado que este tipo de peticiones globales se realiza por bloque, un bajo acierto en la cache local se puede transformar en la situación anteriormente descrita.

Timeout por peticiones GC CR MULTI BLOCK

Una de los problemas más comunes es la perdida de rendimiento en la comunicación UDP IPC de InterConnect del clúster durante determinadas operaciones que ocasionan que las peticiones CR multibloques deban generarse de nuevo.

En determinadas ocasiones, un tamaño insuficiente de buffer de recepción UDP puede causar el overflow y la pérdida de paquetes. En otros casos, una mala configuración o una interfaz defectuosa, por ejemplo, el uso de la interfaz pública

para el tráfico entre los nodos, puede producir una degradación, primero de la red, y después del sistema en general.

El tamaño del buffer de estrada de un socket UDP es establecido al valor 128Kbytes si el sistema operativo por defecto le asigna un tamaño menor y siempre y cuando no se supere el límite establecido por éste.

Es factor es clave si el tamaño de bloques es superior a 8Kbytes y si la plataforma de sistema operativo tiene un valor máximo bajo, como el caso de Linux. En este caso se recomienda realizar un ajuste de la configuración UDP IPC.

Esta es una de las razones por la que en Oracle RDBMS 9iR2 RAC, una analyze schema se ejecuta un 40% más lento que en una base de datos single-instance.

Para localizar este tipo de situaciones se recomienda el uso de SQL_TRACE y/o del evento 10046, además de la vista V$SYSTEM_EVENT para controlar la ratio de timeouts frente a las esperas por las peticiones CR.

Finalmente, y de forma adicional al ajuste de la configuración del tamaño del buffer de entrada UDP, reduciendo el valor del parámetro DB_FILE_MULTI_BLOCK_READ_COUNT a valores de 8 o 4 pueden aliviar e incluso eliminar este problema.

Este parámetro de puede modificar en tiempo real tanto a nivel de sesión como a nivel de sistema. Por tanto, se recomienda establecer dicho parámetro a un valor adecuado para reducir o eliminar esta casuística.

Esperas por segmentos UNDO

Envio excesivo de bloques de UNDO y contención por UNDO buffers

Cuando una sentencia SQL necesita leer un bloque con una transacción activa, tiene que realizar el cleanout del bloque y generar la información de UNDO para crear una versión del CR. Si las transacciones activas pertenecen a mas de una instancia, es necesario combinar la información de UNDO local y remota para realizar un lectura consistente.

Esto provocará solicitudes de tipo CR sobre bloques de UNDO. Dependiendo de la cantidad de bloques que cambian y la duración de las transacciones, este envio de bloques de UNDO puede convertirse en un cuello de botella y afectar al rendimiento del sistema.

Esta situación puede aparecer en sistemas donde las aplicaciones lean datos recientemente insertado con mucha frecuencia y donde la operaciones de commit se dilaten en el tiempo o sean pocos frecuentes.

Este tipo de carga afecta a entornos de bases de datos de una única instancia, pero en el caso de bases de datos en RAC, este problema se agrava claramete dado que un subconkjunto de esta información de UNDO debe ser obtenida desde instancias remotas.

Un ejemplo de este tipo de situación en un entorno de dos instancias, A y B, es la siguiente:

- Alguna de las instancias modifican algún bloque.

- La instancia B consulta uno de esos bloques.
- Esta lectura causa una petición para generar una versión CR de dicho bloque, bien en la instancia local o en la instancia remota, por ejemplo, la instancia A.
- Si es la instancia remota, el proceso LMS de esa instancia A intentará genera el bloque CR.
- Si es la local, instancia B, el proceso que genera la lectura, genera el bloque CR. En este caso, la instancia necesita leer la tabla de transacciones y los bloques de UNDO del segmento de rollback referenciada en la ITL activa del bloque.
- Todo este proceso, la comunicación entre la instancia local y la remota, provoca peticiones CR globales (global CR requests) y dado que las cabeceras de los segmentos de UNDO son frecuentemente accedidas, también pueden aparecer eventos de tipo Buffer Busy Waits.

Estos eventos de espera serán de tipo GC BUFFER BUSY y junto con el uso de la gestión automática de segmentos de UNDO, puede significar un problema de rendimiento ya que cada transacción de la base de datos usará un segmento de UNDO distinto.

Las vistas V$SESSION_WAIT y V$SESSION_WAIT_HISTORY permiten identificar los segmentos de rollback implicados a través de las columnas P1 y P2 que identifican el identificador del fichero y el identificador del bloque respectivamente.

Esta situación de esperas frecuentes por cabeceras y bloques de UNDO se ha aliviado considerablemente en Oracle RDBMS Server 10gR2 y versiones superiores a través de la característica RowCR, que permite generar estas peticiones a nivel de fila y no a nivel de bloque, reduciendo así la contención por este tipo de situaciones.

Esta característica no está habilitada por defecto en Oracle RDBMS Server 10gR2 y para su activación se recomienda contactar con los servicios de soporte de Oracle antes de activar el parámetro _Row_cr=TRUE.

Adicionalmente a esta optimización, la afinidad de UNDO permite reducir el número de peticiones CR, ya que el máster de cualquier bloque o cabecera de UNDO es siempre la instancia que realiza la adquisición del segmento de UNDO.

Por tanto, se eliminan las peticiones de tipo 3-WAY UNDO y por tanto el tráfico entre los nodos del clúster al respecto.

Los sintomas de este tipo de problemas no son fácilmente indentificables desde un informe de Statspack en versiones anteriores a Oracle RDBMS 10g. El evento de espera principal suele ser 'global cache cr request' y 'buffer busy global CR' en el caso de Oracle RDBMS 9i o 'gc buffer busy' y 'gc cr block 2/3 way' en el caso de 10g, pero no hay un forma sencilla de conocer si los bufferes implicados se corresponden o no con bufferes de tipo UNDO.

La vista V$SEGMENT_STATISTICS no contiene informaciñon sobre segmentos de UNDO, por lo que la sección "Top CR Blocks Served per Segment" Statspack no aporta ayuda alguna.

Algunas estadísticas almacenadas en la vista V$SYSSTAT son útiles para medir cuanta información de rollback es generada para lecturas consistentes, como

"cleanouts and rollbacks – consistent read" and "cleanouts only – consistent read". Sin embargo, estas estadísticas no permiten identificar si lo cleanouts y la información de rollback es por bloques remotos de UNDO.

La forma más sencilla de determinar si son peticiones CR sobre bloques de UNDO, es consultar la vista V$CR_BLOCK_SERVER (o X$KCLCRST). En ella, la columna CURRENT_REQUESTS representa el numero de peticiones sobre bloques de UNDO, mientras que la columna CR_REQUESTS representa el número de peticiones sobre bloques de tablas e índices.

En la versión Oracle RDBMS 10g, las estadísticas de la vista V$CR_BLOCK_SERVER están incluidas en los informes de Automatic Workload Repository (AWR).

Como regla general se puede decir que, si el numero de CURRENT_REQUESTS es mayor en el número de CR_REQUESTS, es necesaria una revisión, especialmente si existe un número significativo de buffer busy waits sobre CR.

Para determinar si los busy buffers se corresponden con bloque de UNDO o bufferes de datos, la herramienta mas precisa es la activación del evento 10046 a nivel 8.

Una vez confirmada la situación, el siguiente paso es encontrar los segmentos de índices con transacciones pendientes. Esta tarea no es trivial, ya que no se puede establacer una correlación clara. Los segmentos de índices indicados por AWR pueden ser candidatos, pero puede haber situación de concurrencia en escritura que también provoquen estos cambios.

A partir del estudio de las trazas SQL a nivel 8, pueden extraerse las sentencias SQL implicadas, y des sus planes de ejecuciones, determinar los indices implicados.

Esperas por bloqueos en la Library Cache

Las esperas por Library Cache Lock son de las más comunes y sobre todo en entornos de Oracle Aplications y suelen aparecer entre las 5 primeras causas en aquellas aplicaciones que hacen uso intensivo de PL/SQL y/o Oracle Advanced Queueing.

Este tipo de acceso al diccionario se realiza de forma exclusiva cuando, por ejemplo, es necesario recompilar un procedimiento almacenado o el cuerpo de un paquete PL/SQL.

Por ejemplo, cuando se recompila el cuerpo de un paquete se realizan varios bloqueos de tipo LB en modo compartido, share mode, y además se realiza un bloqueo de tipo LC en modo exclusivo. Este tipo de bloqueos se realizan principalmente en la fase de parsing del código PL/SQL protegen todos los objetos de los que el bloque de código depende.

Por tanto, la recomendación es clara, se deben de minimizar la re compilación de paquetes, procedimientos y funciones PL/SQL durante la actividad de usuarios en la base de datos ya que los mecanismos de protección pueden provocar situaciones de bloqueos y de bajo rendimiento.

En el caso de una configuración en RAC, este tipo de bloqueos se convierte en bloqueos globales en todo el clúster y se deberá mantener la localización de todo objeto en cuanto a su masterización. Si el recurso está masterizado en una de las

instancias remotas, se generará un mensaje de tipo GES para gestionar el acceso al objeto.

En el caso de que este tipo de situaciones sean frecuentes, los mensajes GES para dicha gestión pueden provocar latencias en el acceso a los objetos del diccionario a dilatar la obtención de bloqueos de tipo Library Cache.

Para monitorizar el comportamiento de los bloqueos Library Cache, se recomienda el uso del evento 10706 tanto a nivel de sesión como a nivel de sistema.

Recomendaciones de conectividad a entornos RAC

Balanceo de carga

Connection Load Balancing

Oracle Net Services proporciona la capacidad de distribuir conexiones de clientes a través de las instancias en una configuración de Oracle RAC.

Hay dos tipos de balanceo de carga que puede implementar: balanceo de carga del lado del cliente (*client-side*) y del lado del servidor (*server-side*). El load balancing client-side distribuye las solicitudes de conexión entre los listeners, independientemente de cada cliente. Con el load balancing server-side, el SCAN listener dirige una solicitud de conexión a la mejor instancia que actualmente proporciona el servicio, en función de las configuraciones *-clbgoal* y *-rlbgoal* para el servicio.

El SCAN listener conoce el protocolo HTTP para que pueda redirigir a los clientes HTTP al handler apropiado, que puede residir en diferentes nodos del clúster y no sólo en el nodo en el que reside el SCAN listener que ha recibido la solicitud.

En una base de datos Oracle RAC, las conexiones de los clientes deben usar ambos tipos de conexión balanceada (client-side y server-side).

SERVER-SIDE

Cuando se crea una base de datos Oracle RAC con el dbca, automáticamente configura y habilita el load balancing server-side y crea un ejemplo de definición de conexión client-side load balancing en el tnsnames.ora.

El Oracle Clusterware Database Agent es el responsable de administrar el parámetro LISTENER_NETWORKS.

Si se indica el parámetro de base de datos REMOTE_LISTENER manualmente, hay que configurarlo a `scan_name:scan_port`

FAN, Fast Connection Failover y el Load Balancing Advisory dependen de una configuración precisa de conexión balanceada que incluye establecer el objetivo de balanceo de carga de la conexión para el servicio. Se puede usar un objetivo LONG o SHORT para load balancing de conexiones:

- **LONG**

Se recomienda el uso de LONG cuando se esperan conexiones desde el aplicativo de larga vida, es decir, se espera que permanezcan mucho tiempo conectadas al clúster. Es la configuración que suele emplearse en aplicativos de tipo cliente/servidor o en servidores de aplicaciones, además de ser el valor por defecto.

Un ejemplo de la configuración basada en LONG para el servicio batchconn:

```
$ srvctl modify service -db db_unique_name -service
batchconn -clbgoal LONG;
```

- **SHORT**

Se recomienda el uso de SHORT cuando se esperan conexiones desde el aplicativo de sean cortas, es decir, se espera que el aplicativo se conecte, realice alguna operación y se desconecte. Es el valor que suele recomendarse para aplicativos donde no existe pools de conexiones.

Un ejemplo de la configuración basada en SHORT para el servicio oltpapp:

```
$ srvctl modify service -db db_unique_name -service
oltpapp -clbgoal SHORT;
```

CLIENT-SIDE

El balanceo de carga client-side se indica en la definición de conexión del cliente (archivo tnsnames.ora comúnmente) configurando el parámetro LOAD_BALANCE=ON. Cuando establece este parámetro en ON, Oracle Database selecciona aleatoriamente una dirección en la lista de direcciones y se conecta al listener de ese nodo. Esto equilibra las conexiones de cliente a través de los listener de SCAN disponibles en el clúster.

En caso de que se configure la SCAN en la solicitud de conexión del cliente (altamente recomendado), no es necesario usar LOAD_BALANCE=on ya que Oracle Net balancea automáticamente la carga de solicitudes de conexión de cliente a través de las tres direcciones IP que se definió para la SCAN (a menos que esté usando EZConnect). Es decir, cuando los clientes se conectan usando SCAN, Oracle Net balancea automáticamente las peticiones de conexión de cliente a través de las tres IPs de SCAN.

El listener de SCAN redirige la solicitud de conexión al listener local de la instancia que está menos cargada (si -clbgoal está configurado en SHORT) y proporciona el servicio solicitado. Cuando el listener recibe la solicitud de conexión, el listener conecta al usuario con una instancia que el listener sabe que proporciona el servicio solicitado. Para ver qué servicios admite un listener, se puede ejecutar el comando `lsnrctl services`.

Para poder usar SCAN en clientes que no lo soportan, por ejemplo clientes versiones anteriores a 11gR2, hay que cambiar el tnsnames.ora del cliente, indicando todas las SCAN VIPs y el LOAD_BALANCE=on.

Si la base de datos no es 11gR2 o superior y se desea usar SCAN, hay que indicar las SCAN VIPs en el parámetro REMOTE_LISTENER.

Además del client-side load balancing, en Oracle 12cR2 Net Services se sigue incluyendo el connection failover para clientes antiguos. Si se devuelve un error de la dirección elegida en la lista, Oracle Net Services intenta la siguiente dirección en la lista hasta que sea exitosa o haya agotado todas las direcciones. Para SCAN, Oracle Net Services prueba las tres direcciones antes de devolver una fallo al cliente.

Load Balancing Advisory

Desde la versión Oracle 10gR2, la gestión del balanceo de carga puede realizarse a través del Load Balancing Advisory (LBA).

Oracle recomienda que las aplicaciones utilicen pools de conexión con conexiones persistentes que abarquen las instancias que ofrecen un servicio en particular. Al usar conexiones persistentes, las conexiones se crean con poca frecuencia y existen durante un período prolongado. El trabajo entra al sistema con alta frecuencia, toma prestadas estas conexiones y existe durante un tiempo relativamente corto. LBA proporciona consejos sobre cómo dirigir el trabajo entrante a las instancias que proporcionan la calidad de servicio óptima para ese trabajo. Esto minimiza la necesidad de reubicar el trabajo más tarde.

LBA se implementa con los clientes clave de Oracle, como un listener, un JDBC universal connection pool, OCI session pool, Oracle WebLogic Server Active GridLink for Oracle RAC y ODP.NET Connection Pools. Las aplicaciones de terceros también pueden suscribirse a eventos de LBA mediante JDBC y la API de Oracle RAC FAN o mediante el uso de callbacks con OCI.

Puede configurar su entorno para utilizar LBA definiendo objetivos de nivel de servicio para cada servicio de base de datos para el que desee habilitar el balanceo de carga. Las solicitudes de conexión se puedan enrutar en base a dos métricas, SERVICE TIME y THROUGHPUT.

Finalmente, es posible establecer adicionalmente el valor NONE como objetivo, con lo que se deshabilitaría el balanceo de carga.

Para comprobar la configuración para un determinado SERVICE_NAME pueden consultarse las vistas DBA_SERVICES, V$SERVICES y V$ACTIVE_SERVICES.

SERVICE TIME

Intenta dirigir las peticiones hacia las instancias del clúster en función del tiempo de respuesta. Para ello LBA, se basa en estadísticas de elapse time de ejecución así como en el ancho de banda disponible.

Ejemplo de configuración para conexiones basadas en SERVICE_TIME usando el servicio db_servicio:

```
$ srvctl modify service -db db_unique_name -service
db_servicio -rlbgoal SERVICE_TIME -clbgoal SHORT
```

THROUGHPUT

Intenta dirigir las peticiones hacia las instancias del clúster en función del rendimiento concurrente. Para ello LBA, se basa en estadísticas de elapse time de ejecución así como en el ancho de banda disponible.

Ejemplo de configuración para conexiones basadas en THROUGHPUT usando el servicio db_servicio:

```
$ srvctl modify service -db db_unique_name -service
db_servicio -rlbgoal THROUGHPUT -clbgoal LONG
```

Grid Infrastructure Single Client Access Name (SCAN)

Oracle Clusterware puede usar el Single Client Access Name (SCAN) para la configuración dinámica de direcciones VIP, eliminando la necesidad de realizar la configuración de servidor de forma manual.

SCAN es un nombre de dominio registrado en al menos una y hasta tres direcciones IP, ya sea en DNS o GNS con DHCP. Se recomiendan 3 direcciones IP y siempre que sea posible, configurarlo en DNS.

Se debe configurar lo siguiente:

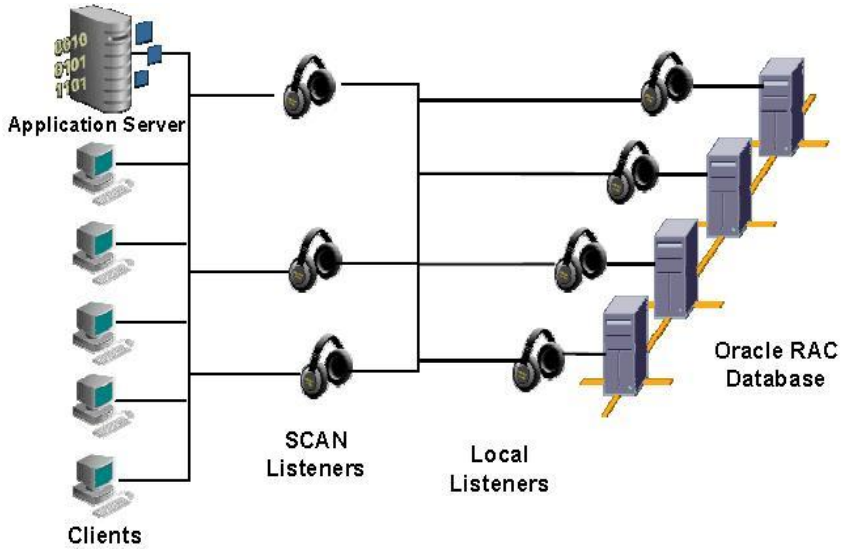
- Una dirección pública y nombre de host para cada nodo del cluster.
- Una dirección VIP para cada nodo. Debe asignar una dirección VIP a cada nodo del clúster. Cada dirección VIP debe estar en la misma subred que la dirección IP pública del nodo y debe ser una dirección a la que se le asigne un nombre en el DNS. Estas VIPs no deben estar en uso desde dentro de la red antes de instalar Oracle Grid Infrastructure.
- Un SCAN name es el hostname virtual, registrado con hasta tres direcciones IP de SCAN para el clúster. Para una alta disponibilidad y escalabilidad, Oracle recomienda que se configuren 3 IPs de SCAN

A tener en cuenta:

- SCAN es una parte elemental de Grid Infrastructure desde 11gR2 y no está soportado eliminarlo. Por lo que debe ser el método de conexión utilizado.
- Oracle recomienda que todas las conexiones a las bases de datos Oracle RAC (y single-instances gestionadas con Clusterware) usen SCAN en su cadena de conexión de cliente.
- Con SCAN, no tiene que cambiar la cadena de conexión del cliente ni siquiera cuando la configuración del clúster cambia (nodos añadidos o eliminados).
- El registro del SCAN name se debe hacer en DNS o GNS, no se puede usar el /etc/hosts para resolver la SCAN
- En Grid Infrastructure 11gR2 y 12cR1, en una instalación "Typical", el SCAN name se correspondía con el cluster name.
- Si el SCAN name y el nombre del clúster se ingresan en el mismo durante la instalación, los requisitos del nombre del clúster se aplican al SCAN name.
- En Grid Infrastructure 11gR2 y 12cR1, en la instalación "Advanced", se indican en campos separados. Y a partir de Grid Infrastructure 12cR2, el SCAN name y el nombre del clúster se ingresan en campos separados durante la instalación.
- El nombre del clúster "cluster name" no distingue entre mayúsculas y minúsculas (es case-insensitive), debe ser único en la organización, debe tener al menos un carácter de longitud y no más de 15 caracteres, debe ser alfanumérico, no puede comenzar con un número y puede contener guiones (-), pero los caracteres de subrayado (_) no están permitidos.
- Seleccione el nombre de clúster cuidadosamente. Después de la instalación, solo puede cambiar el nombre del clúster reinstalando Oracle Grid Infrastructure.
- Si el SCAN name y el nombre del clúster se ingresan en campos separados durante la instalación, los requisitos del nombre del clúster no se aplican al SCAN name, que puede tener más de 15 caracteres.
- Desde Oracle Database 12cR2, los SCAN listeners soportan el protocolo Http
- Desde Grid Infrastructure 12cR1, se soporta IPv6 en la red pública; esto incluye SCAN, IPs y VIPs públicas

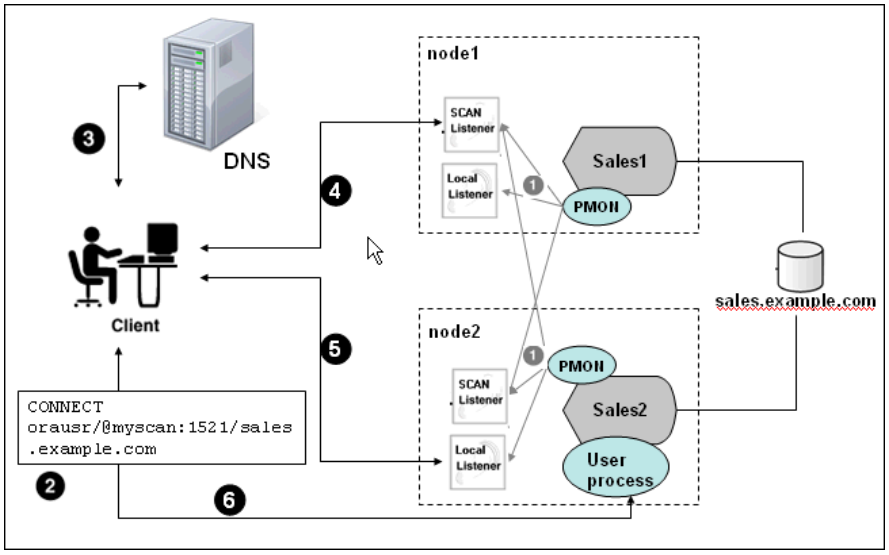
- Desde Grid Infrastructure 12cR1, SCAN admite varias subredes en el clúster (una SCAN por subred)

Cómo se realizan las conexiones usando SCAN:



Cuando un cliente envía una petición de conexión al SCAN name, este resuelve por DNS (o GNS con DHCP) a una de las 3 IPs de SCAN, que contacta con su SCAN listener que está escuchando en su SCAN VIP por el puerto especificado (SCAN port). Como todos los servicios en el cluster están registrados con el SCAN listener, el SCAN listener responde al cliente con la dirección del local listener de la instancia y nodo con menos carga (cada SCAN listener mantiene las estadísticas de carga del cluster actualizadas) de los que ofrecen actualmente el servicio de BBDD. El cliente conecta con el local listener indicado, quien inicia un proceso de servidor dedicado para la conexión a la base de datos. Finalmente, el cliente conecta directamente con el proceso de servidor dedicado y accede a la base de datos por la instancia que se está ejecutando en el nodo del local listener devuelto. Todas estas operaciones son transparentes para el cliente, sin necesidad de realizar ninguna configuración explícita.

Load Balancing en Grid Infrastructure usando Single Client Access Name (SCAN):



1.- El proceso PMON de cada instancia registra el servicio de base de datos con el listener por defecto del nodo local y con cada SCAN listener, que está especificado por el parámetro de base de datos REMOTE_LISTENER. Los listeners son dinámicamente actualizados de la cantidad de trabajo que está siendo manejado por las instancias y los dispatchers.

2.- El cliente realiza una conexión usando el descriptor de la conexión con: usuario/passwd, nombre de SCAN, puerto y nombre de la BBDD:

```
SQL> connect orausr/@scan_name:1521/sales.example.com
bash$ sqlplus orausr@sales (sqlnet.ora)
```

3.- El cliente consulta al DNS para resolver el scan_name. El nombre de SCAN estará asociado normalmente a 3 IP's en DNS con round-robin. El DNS devuelve al cliente esas 3 direcciones asignadas al scan_name, el cliente envía una petición de conexión a la primera dirección IP. Si la petición de conexión falla, entonces el cliente envía una nueva petición de conexión usando la siguiente dirección IP.

4.- Cuando la petición de conexión es satisfactoria, el cliente conecta con el SCAN listener del cluster que contiene la base de datos “sales” en este ejemplo. El SCAN listener compara la carga de trabajo de las instancias sales1 y sales2 de la base de datos sales y la carga de trabajo de los nodos en los que se ejecuta. En este ejemplo, el node2 tiene menos carga que el node1, por lo que el SCAN listener selecciona el node2 y envía la dirección del local listener de ese nodo (node2) de vuelta al cliente.

5.- El cliente conecta con el local listener del node2, que está escuchando en la VIP local y puerto especificado. El local listener inicia un proceso de servidor dedicado para la conexión a la base de datos.

6.- El cliente conecta directamente con el proceso de servidor dedicado del node2 y accede a la instancia de base de datos sales2.

Comportamiento de las VIP

Las "node VIP" o también llamadas "local VIP" o simplemente VIP son las IPs virtuales de los nodos. Oracle Clusterware/Real Application Cluster aloja las "node VIP" en la red pública. Las VIP de nodo son las direcciones que usan los clientes para conectarse a una base de datos Oracle RAC.

En el sub-apartado anterior hemos visto cómo se realizan las conexiones típicas de un cliente de base de datos a una instancia de base de datos Oracle RAC, usando SCAN que es el método recomendado.

Las node VIP de RAC están diseñadas para trabajar en nodos del clúster donde existen instancias de bases de datos activas y tan solo aceptan conexiones cuando están activas en nodo donde fueron asignadas y que llamaremos HOME NODE.

Si un nodo falla, Oracle Clusterware conmuta su dirección VIP a otro nodo del cluster disponible en el que la dirección VIP pueda aceptar conexiones TCP, pero no acepta conexiones a la base de datos Oracle.

Los clientes que intentan conectarse a una VIP que no reside en su HOME NODE reciben un error de rechazo de conexión rápida en lugar de esperar un timeout de conexión TCP. El cliente intentará inmediatamente una conexión a la siguiente dirección de la lista o SCAN VIP.

Cuando la red en la que se configura la VIP vuelve a estar en línea, Oracle Clusterware devuelve la VIP a su HOME NODE, donde se aceptan las conexiones.

En general, las direcciones VIP fallan cuando:

- El nodo en el que se ejecuta una dirección VIP falla
- Todas las interfaces de la dirección VIP fallan
- Todas las interfaces de la dirección VIP están desconectadas de la red

La recomendación general es usar SCAN con database Services para redirigir los clientes a los diferentes nodos del clúster, ya que pueden ser dinámicamente realojados y modificados en cualquier momento.

Transacciones distribuidas (XA)

Oracle RAC posee una serie de restricciones o best practices respecto al uso de transacciones distribuidas o XA. Estas restricciones difieren en función de las versiones y han evolucionado desde las primeras versiones de OPS/RAC. A continuación se muestran las más importantes:

- Hasta la versión 11gR1 de Oracle RAC, la totalidad de los hilos de la transacción distribuida debía ser ejecutada en la misma instancia de la base de datos. En caso de failover y de la versión de Oracle RAC, se daban escenarios diferentes.

- Hasta la versión 11gR1, una transacción XA no podía ser balanceada entre las diferentes instancias. La especificación XA permite que los hilos/brazos de la transacción se suspendan y sean continuados por el Transaction Manager (TM). En el caso de usar balanceo de carga, cualquier transacción al ser continuada puede hacerlo en una instancia distinta a la que inicio el hilo/brazo de la transacción. A partir de Oracle RAC 11gR1, esta limitación fue eliminada por defecto, sin embargo sigue existiendo implicaciones por impacto en el rendimiento de este tipo de transacciones.

- No existe una forma fiable y segura de garantizar la unicidad del identificador glabla de la transacción (XID) dentro de Oracle RAC. Oracle RAC no comprueba que el XID proporcionado es única a lo largo de todas las instancias del clúster. El TM necesita mantener esta unicidad de los XID.

- Hasta la versión 11gR1, para evitar la posibilidad de pérdida de visibilidad de los cambios realizados por transacciones distribuidas fuertemente acopladas, todas los hilos/brazos de las transacciones debían localizarse en la misma instancia. Aunque es una recomendación, y no una obligatoriedad, es muy recomendable localizar todos los hilos/brazos también de las transacciones débilmente acopladas. A partir de 11gR1, está limitación fue eliminada, y todas las transacciones distribuidas se tratan como fuertemente acopladas, de ahí que tenga implicaciones en el rendimiento del sistema en caso de usarse contra diferentes instancias.

- Si por la razón que sea, por ejemplo, perdida de conexión entre el TM y el Resource Manager (RM), el TM intenta obtener el estado de la transacción global en una instancia distinta del clúster, puede llegar a obtener valores incorrectos hasta que bien los segmentos de UNDO de la instancia fallida sean recuperados o bien todas las transacciones de la instancia original agoten su tiempo de espera o time-out. Si las peticiones de estado del TM se realizan sobre una instancia distinta a la instancia donde se inició, la respuesta más probable será el error "ORA-24756: Transaction does not exist" hasta que la transacción agote su tiempo de espera, pasando a dudosa o in-doubt o bien siendo recuperada como parte del proceso de recover de la propia instancia.

- La situación descrita anterirormente, errores ORA-24756, es común en entornos basados en Oracle Tuxedo ya que puede hacer uso de xa_start (resume). En este caso las opciones para evitar estos errores son las siguientes:

    - No realizar este tipo de operaciones xa_start, algo que no siempre es posible.

    - Aplicar la misma configuración con servicios Distributed Transaction Processing (DTP) de Oracle 10gR2 que se describen en el siguiente apartado.

Oracle 10gR2 RAC

En Oracle RAC 10gR2, se introduce una nueva funcionalidad para soportar el procesamiento de transacciones distribuidas, Distributed Transaction Processing (DTP). DTP permite de la recuperación de transacciones XA dentro de una base de datos en RAC de una forma óptima.

Durante el procesamiento normal de un transacciones y concretamente en la fase de prepare, se genera la información de UNDO. Si ésta falla, la información genera es almacenada en tablas de sistema que posteriormente permitirá la realización de un proceso de recovery externo. Durante las operaciones, esta información es almacenada de forma no acoplada al procesamiento de las transacciones, modo lazy. Sin embargo, esto implica que hasta que todas las transacciones en vuelo en modo prepared estén en dichas tablas internas, el propio servicio DTP no podrá re arrancarse en cualquiera de las instancias supervivientes.

En 10gR2 el requisito de realizar siempre una operación xa_recover sobre la instancia fallida puede relajarse en el caso de que el TM no tenga información suficiente sobre la transacción en vuelo para realizar la recuperación de la transacción usando xa_commit o xa_rollback.

Debido a esto y desde la llegada de los servicios DTP, el procedimiento interno de Oracle, SYS.DBMS_SYSTEM.DIST_TXN_SYNC realmente realice una operación nula o no-op.

Sin embargo, no todas las aplicaciones usan este tipo de servicios, por lo que a partir de 10.2.0.3 y a través del patch#5968965, se vuelve a habilitar dicho método SYS.DBMS_SYSTEM.DIST_TXN_SYNC.

A partir de 10.2.0.4, el método SYS.DBMS_SYSTEM.DIST_TXN_SYN se renombra a DBMS_XA.DIST_TXN_SYNC como medida de seguridad y por tanto no es necesario otorgar privilegios a las aplicaciones sobre dicho paquete. Este hecho es importante, porque significa un cambio necesario a nivel de clientes, éstos tiene que invocar el paquete DBMS_XA en vez de al paquete DBMS_SYSTEM.

Finalmente, en 10gR2 el parámetro MAX_COMMIT_PROPAGATION_DELAY queda como "deprecated" y solo es mantenido por compatibilidad con versiones anteriores de Oracle RAC. En su valor por defecto, 0, causa el refresco inmediato del SCN después de cada commit, broadcast commit, permitiendo que cualquier dato insertado o modificado está inmediatamente disponible en cualquier instancia del clúster.

Desde Oracle 11gR1 RAC

En 11gR1se introduce la capacidad de proceso de transacciones globales a través del clúster, permitiendo así, las transacciones distribuidas pueda procesarse en cualquier instancia del éste. Concretamente se añaden las siguientes características:

- Un nuevo tipo de enqueue, global DX (Distributed Transaction) para transacciones fuertemente acopladas.
- Consistencia en lectura para las transacciones fuertemente acopladas, por ejemplo, pre-commit de cambios realizados entre instancias.
- Se crea un nuevo proceso de tipo background, GTXn, manejado a través del parámetro de iniciación GLOBAL_TXN_PROCESSES para la gestión de los enqueue de tipo DX y la realización de procesos de 2PC entre las instancias.
    - El número de GLOBAL_TXN_PROCESSES es automáticamente gestionado por la propia base de datos.
    - Sin embargo, están documentados comportamientos del autotuning de este parámetro que impacta en el rendimiento por la aparición de eventos de tipo "Global transaction acquire ins".
    - En este caso, y tras confirmación de los servicios de soporte de Oracle, puede habilitarse el parámetro _DISABLE_AUTOTUNE_GTX para deshabilitar las funciones de auto tuning.

Finalmente, la gestión de global de transacciones distribuidas en todo el clúster es una funcionalidad por defecto de Oracle 11gR1 RAC habilitada a través de un valor del parámetro COMPATIBLE >= 11.0 y solo puede ser deshabilitada a través del parámetro no documentado _CLUSTERWIDE_GLOBAL_TRANSACTIONS con un valor de FALSE. En este caso, el mecanismo de procesamiento de transacciones distribuidas es el equivalente a 10gR2.

Configuración de los servicios SQL*Net

El fichero de tnsnames.ora es creado en cada uno de los clientes de las bases de datos. Cada entrada en este fichero equivale a un identificador de conexión que

incluye la ruta de red contra un servicio que se encuentra disponible en el servidor de base de datos.

Tipo de servicio de red	Descripción

Conexión de base de datos	Clientes que conectan a cualquier instancia usando la entrada de servicios de red.
	Ejemplo recomendado antes de 11gR2 o sin SCAN:
	`db.example.com=`
	`(description=`
	`(load_balance=on)`
	`(address=(protocol=tcp)(host=node1-vip)(port=1521))`
	`(address=(protocol=tcp)(host=node2-vip)(port=1521))`
	`(connect_data=(service_name=db.example.com)))`
	Ejemplo recomendado desde 11gR2 con SCAN:
	`db.example.com=`
	`(description=`
	`(address=(protocol=tcp)(host= scan-name-rac.example.com)(port=1521))`
	`(connect_data=(service_name=db.example.com)))`

Conexión a instancias	Por defecto la conexión se realizara a conexión de la base de datos tal y como se comenta en la entrada anterior, pero si por algún motivo se permite la conexión directa a una instancia directamente, se debe realizar de la siguiente forma:
	`db1.example.com=`
	`(description=`
	`(address=(protocol=tcp)(host=node1-vip)(port=1521))`
	`(connect_data=`
	`(service_name=db.example.com)`
	`(instance_name=db1)))`

Servicios	Cuando se configura servicios de alta disponibilidad:
	Ejemplo recomendado antes de 11gR2 o sin SCAN:
	`db_servicio =`
	`(description =`
	`(address=(protocol=tcp)(host=node1-vip)(port=1521))`
	`(address=(protocol=tcp)(host=node2-vip)(port=1521))`
	`(load_balance=yes)`
	`(connect_data=`
	`(server = dedicated)`
	`(service_name = servicio.example.com)`
	`)))`
	Ejemplo recomendado desde 11gR2 con SCAN:
	`db_servicio =`
	`(description=`
	`(address=(protocol=tcp)(host= scan-name-rac.example.com)(port=1521))`
	`(connect_data=(service_name=db_servicio)))`

Tecnologías de conectividad a entornos RAC

Oracle WebLogic Server 12c Active GridLink (AGL)

En servidores de aplicaciones de tipo JEE, las interacciones con la base de datos son manejadas normalmente a través de data source (DS) JDBC.

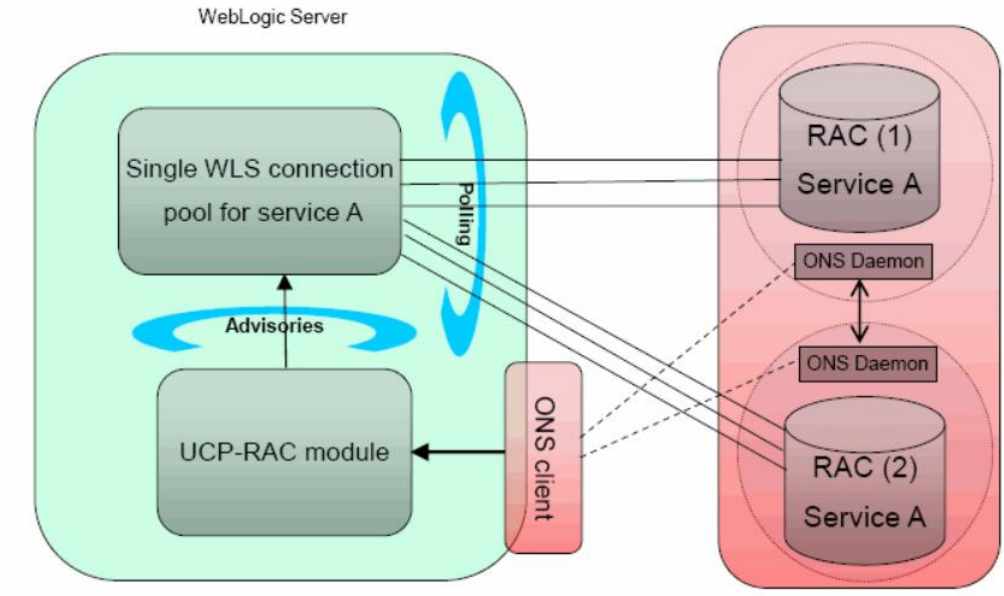
En Oracle WebLogic Server (WLS), se configura la conectividad con la base de datos mediante la configuración de DS y éste de tipo GridLink, desplegando éstos como recursos JDBC en los distintos servidores y/o clústeres en un dominio de WebLogic.

Cada DS que se configura contiene un pool de conexiones a una base de datos. Estos DS se crean en su proceso de creación, en el despliegue o targeting de la instancia del DS en un servidor y/ clúster así como en el proceso de arranque del servidor.

Un único origen de datos AGL proporciona conectividad entre WebLogic Server y un servicio de Oracle Database, que puede incluir uno o más clústeres de Oracle RAC.

Oracle no recomienda el uso de datasources genéricos con clústeres de Oracle RAC.

Las aplicaciones usan un DS GridLink a través de JNDI o en el contexto local de la aplicación (java:comp/env) de la misma forma que con un DS tradicional y este es el que realmente solicita la conexión a base de datos.



Un origen de datos AGL incluye las características de fuentes de datos genéricas más el siguiente soporte para Oracle RAC:

- Fast Connection Failover
- Runtime Connection Load Balancing
- GridLink Affinity
- SCAN Addresses

- Secure Communication using Oracle Wallet with ONS Listener.

El alcance de este documento no se centrara en el análisis de las características anteriores citadas, sino en desarrollar la correcta configuración y requerimientos para el uso de Active GridLink como configuración estándar de datasources contra base de datos en Real Application Cluster.

Para crear un origen de datos AGL en su dominio WebLogic, puede utilizar la Consola de administración de WebLogic Server o la Herramienta de comandos WebLogic (WLST).

Los siguientes apartados proporcionan una descripción general de los pasos básicos utilizados en el asistente de configuración de origen de datos para crear un origen de datos mediante la Consola de administración de WebLogic Server:

- JDBC Data Source Properties
- Configure Transaction Options
- Configure Connection Properties
- Test Connections
- ONS Client Configuration
- Test ONS Client Configuration
- Target the Data Source

Oracle WebLogic Server Multi Datasources (MDS)

En servidores de aplicaciones de tipo JEE, las interacciones con la base de datos son manejadas normalmente a través de data source (DS) JDBC.

En Oracle WebLogic Server (WLS), se configura la conectividad con la base de datos mediante la configuración de DS y multi data sources (MDS) JDBC, desplegando éstos como recursos JDBC en los distintos servidores y/o clústeres en un dominio de WebLogic.

Cada DS que se configura contiene un pool de conexiones a una base de datos. Estos DS se crean en su proceso de creación, en el despliegue o targeting de la instancia del DS en un servidor y/ clúster así como en el proceso de arranque del servidor.

Un MDS es una abstracción de un conjunto de DS que proporciona balanceo de carga o funciones de failover en todo el conjunto de DS asociados al MDS.

Las aplicaciones usan un MDS a través de JNDI o en el contexto local de la aplicación (java:comp/env) de la misma forma que con un DS tradicional y éstos son los que realmente solicitan la conexión a base de datos.

Cada MDS determina que DS es el más adecuado para satisfacer las necesidades de la petición en función del algoritmo seleccionado en la configuración del MDS.

Existen dos algoritmos disponibles para los MDS:

- Balanceo de carga: El acceso a los pools en WLS se hace mediante el método de round-robin. Al solicitase una nueva conexión, WLS seleccionará una conexión basada en el siguiente pool de conexiones en el orden especificado.

- Alta disponibilidad: Los pools de conexiones se muestran en el orden que se determina cuando se produce una conmutación o failover en el pool de conexiones. WLS intentará obtener de conexiones de base de datos empezando por el primer pool de conexiones de la lista. Si por alguna razón, ese pool falla, utiliza el siguiente según el orden especificado.

Configurando Multi data sources con o sin transacciones globales

Hay tres consideraciones para determinar cómo debe ser configurado los MDS de WLS para hacer uso de las capacidades y funcionalidades de Oracle RAC.

La tabla siguiente muestra las claves:

¿Balanceo de carga?	¿Failover?	¿XA?	¿JDBC Store?	Recomendación
Si	Si	Si		MDS con soporte XA
Si	Si		Si	MDS sin soporte XA
	Si		Si	Connect time Failover sin XA (1)

(1) No incluida en la arquitectura de referencia, solo con fin informativo.

Multi data sources con transacciones globales

Si tus aplicaciones requieren soporte para transacciones globales al acceder a Oracle RAC se debe considerar el uso de MDS con la opción de configuración de soporte de transacciones distribuidas (XA). Teniendo en cuenta que en caso de failover, será manejado por los MDS en lugar de Oracle RAC.

En un escenario con 2PC, si hay un fallo en una instancia de Oracle RAC antes del PREPARE, la operación se reintenta hasta que expira el tiempo de RETRY. Si hay un fallo después de PREPARE, la transacción se pasa a otra instancia.

Esta opción requiere que todos los DS definidos para el MDS usen un driver XA-enabled, o que ninguno lo tenga. Además, todas las propiedades relacionadas con XA deben tener el mismo valor para todos los DS.

Hay varios atributos que es necesario configurar en los DS, a continuación vemos como:

- Primero, es necesario establecer el tipo de driver, en este caso, el driver será ORACLE JDBC THIN.

```
<url>jdbc:oracle:thin:@lcqsol24:1521:SNRAC1</url>
<drivername>
oracle.jdbc.xa.client.OracleXADataSource</driver-name>
```

- Configurar KeepXAConnTillTxComplete="true", para forzar a los DS a reservar una conexión física a la base de datos y a permanecer en el mismo ámbito de la transacción durante su ciclo de vida para esta aplicación en particular.

- Configurar XARetryDurationSeconds="300", el valor es el tiempo durante el cual el transaction manager de WLS reintentará para llamadas XA de tipo recover, commit o rollback.

- Configurar TestConnectionsOnReserve="true", se usa para verificar las conexiones a la base de datos. Puede ser acompañado con un atributo TestTableName.

Parámetro	Valor
Driver JDBC	Oracle JDBC Thin
KeepXAConnTillTxComplete	TRUE
XARetryDurationSeconds	300
TestConnectionsOnReserve	TRUE

En el siguiente ejemplo, se muestra la configuración parcial que incluye las propiedades requeridas para utilizar MDS con Oracle RAC soportando transacciones globales de tipo XA:

```
<jdbc-driver-params>
   <url>jdbc:oracle:thin:@lcqsol24:1521:SNRAC1</url>
<driver-name>
oracle.jdbc.xa.client.OracleXADataSource</driver-name>
……
</jdbc-driver-params>
   <jdbc-connection-pool-params>
   <test-table-name>SQL SELECT 1 FROM DUAL</test-table-name>
   <profile-type>0</profile-type>
</jdbc-connection-pool-params>
<jdbc-data-source-params>
   <jndi-name>oracleRACXAJndiName</jndi-name>
   <global-transactions-protocol>TwoPhaseCommit
   </global-transactions-protocol>
</jdbc-data-source-params>
<jdbc-xa-params>
   <keep-xa-conn-till-tx-complete>true</keep-xa-conn-tilltx-
   complete>
   <xa-end-only-once>true</xa-end-only-once>
   <xa-set-transaction-timeout>true</xa-set-transactiontimeout>
   <xa-transaction-timeout>120</xa-transaction-timeout>
   <xa-retry-duration-seconds>300</xa-retry-durationseconds>
```

Multi data sources sin transacciones globales

Si las aplicaciones no requieren soporte para transacciones globales de tipo XA, se deben configurar MDS para manejar los procedeos de failover y el balanceo de carga sin especificar las propiedades de configuración XA.

Desde la perspectiva de la configuración, la única diferencia con respecto a la configuración anterior es que no hay que especificar los atributos relacionados con XA.

A continuación encontramos un ejemplo de configuración:

```
<jdbc-driver-params>
   <url>jdbc:oracle:thin:@lcqsol24:1521:snrac1</url>
   <driver-name>oracle.jdbc.OracleDriver</driver-name>
   <properties>
     <property>
       <name>user</name>
       <value>wlsqa</value>
     </property>
   </properties>
   <password-encrypted>{3DES}aP/xScCS8uI=</passwordencrypted>
</jdbc-driver-params>
<jdbc-connection-pool-params>
   <test-connections-on-reserve>true</test-connections-onreserve>
   <test-table-name>SQL SELECT 1 FROM DUAL</test-table-name>
</jdbc-connection-pool-params>
<jdbc-data-source-params>
   <jndi-name>jdbcDataSource</jndi-name>
</jdbc-data-source-params>
```

Configurando Connect-Time Failover (CTF)

Tal como se ha comentado anteriormente, CTF no está incluido en la arquitectura de referencia de WLS ni de Oracle RAC. Sin embargo se recoge en este documento con fines informativos.

El connect-time failover de Oracle RAC se puede usar en aplicaciones para manejar los procesos de failover. Esta opción no soporta balanceo de carga y transacciones globales.

En este caso, cada DS está configurado para apuntar a múltiples nodos de Oracle RAC. En escenarios de fallo, el tiempo de failover en la conexión de la base de datos es tanto como sea el timeout de TCP, configurado a nivel de sistema operativo.

La configuración para hacer uso del connect-time failover se fija en el atributo JDBC driver URL, además, hay que especificar la propiedad ConnectionReserveTimeoutSeconds, para especificar el tiempo que una aplicación debe esperar a que la conexión esté disponible.

A continuación podemos encontrar un ejemplo tenemos un ejemplo:

```
<jdbc-driver-params>
   <url>jdbc:oracle:thin:@(DESCRIPTION=
    (ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)
     HOST=lcqsol24)(PORT=1521))(ADDRESS=(PROTOCOL=TCP)
    (HOST=lcqsol25)(PORT=152))(FAILOVER=on)
    (LOAD_BALANCE=off))(CONNECT_DATA=(SERVER=DEDICATED)
    (SERVICE_NAME=snrac)))</url>
   <driver-name>oracle.jdbc.OracleDriver</driver-name>
</jdbc-driver-params>
<jdbc-connection-pool-params>
   <test-connections-on-reserve>true</test-connections-onreserve>
   <test-table-name>SQL SELECT 1 FROM DUAL</test-table-name>
   <profile-type>4</profile-type>
</jdbc-connection-pool-params>
<jdbc-data-source-params>
   <jndi-name>oracleRACJndiName</jndi-name>
   <global-transactions-protocol>OnePhaseCommit
   </global-transactions-protocol>
</jdbc-data-source-params>
```

Network Failover (NF)

Network failover o *connection failover (NF* o *CF)* es el mecanismo de failover por defecto, el más básico y el único disponible para el uso del driver *JDBC* de tipo *THIN* aunque también está disponible para el *driver JDBC Thick OCI8*.

NF asegura que las nuevas y solo las nuevas conexiones que establecen los servidores de aplicaciones contra una instancia de la base de datos en *RAC* que ha tenido una situación de no-disponibilidad se crean contra una de las instancias de backup o supervivientes de la base de datos en clúster incluso si se usa el alias *TNS* correspondiente a la instancia no disponible.

NF es el único mecanismo de failover donde las conexiones existentes no son automáticamente recreadas en las instancias del *RAC* supervivientes. Estas conexiones no podrán ser usadas en caso de failover y recibirán el error *ORA-03113* tras la primera ejecución tras el failover y el error *ORA-03114* en las sucesivas por parte del propio cliente.

Las conexiones en proceso de conexión durante la no-disponibilidad de la instancia, incluidas las operaciones de *Oracle Advanced Queueing*, pueden fallar con una gran variedad de excepciones dependiente en el momento que se encuentre.

Esta situación conlleva, por ejemplo, que se tenga que reiniciar un contenedor *J2EE* para que las aplicaciones contenidas puedan crear de nuevo el pool de conexiones contra la base de datos o que la aplicación tenga que implementar una funcionalidad de reinicio del *pool* de conexiones de manera interna a la aplicación.

Transparent Application Failover (TAF)

Transparent Application Failover (TAF) usa un mecanismo de failover en tiempo de ejecución creado para entornos de *HA*, como *RAC* y *Oracle DataGuard* que permite el restablecimiento de las conexiones con la base de datos.

Esto permite que las aplicaciones clientes se reconecten automáticamente a la base de datos si la conexión falla y opcionalmente, continuar la sentencia *SQL* o reejecutando la sentencia *DML* o *DDL* que estaban en progreso.

Esta reconexión automática es posible gracias al interfaz *Oracle Call Interface (OCI)* en su versión 8. Por tanto *TAF* solo está disponible en el tipo de *driver JDBC Thick* de tipo *OCI8*, y pasa su activación es necesario configurar el parámetro *FAILOVER_MODE* como parte de la cláusula *CONNECT_DATA* del alias *TNS* usada para la creación de la conexión *JDBC* o bien codificarla mediante la URL de conexión.

TAF posee el mejor mecanismos de failover para las conexiones en proceso de conexión a una instancia que es parte de una base de datos de un clúster RAC. Además asegura que las conexiones existentes y que no están siendo usadas en el momento del failover son reconectadas las instancias de bases de datos supervivientes.

Sin embargo, existe la posibilidad de que *TAF* no pueda reejecutar una operación transaccional realizada desde la última operación de *commit*. Cuando esto ocurra el cliente normalmente eleva el error *ORA-25408 "Cannot safely replay call"* y será responsabilidad de la aplicación realizar explícitamente un *rollback* de la transacción

en proceso antes de poder usar la nueva conexión establecida sobre otra de las instancias de la base de datos.

Es importante dejar claro cuáles son las funcionales esperadas y cuáles son las funcionalidades que no implementa *TAF*, por tanto, el nivel de protección de *TAF* cubre los siguientes elementos:

- Conexiones creadas contra la base de datos.
- Estado de las sesiones de usuario.
- Sentencias en modo *prepared*
- Cursores activos de sentencias *SELECT* que comenzaron a devolver resultados en el momento del fallo.

TAF no protege ni soporta failover en las siguientes situaciones o elementos:

- Aplicaciones que no usen la interfaz *OCI8* ó superior.
- Variables de aplicación de tipo *server-side*, como estados de los paquetes *PL/SQL*.
- Las transacciones activas de sentencias *UPDATE*.

TAF implementa un sistema de mensajería asíncrona basada en eventos que permite a la base de datos comunicarse con los clientes que estén usando este tipo de conexión..

Los diferentes eventos TAF se documentan en el apéndice A de este mismo documento.

El aplicativo debe capturar estos eventos e implementar la respuesta apropiada en función de la lógica de negocio.

Esta captura de eventos se realiza mediante una función callback que la aplicación debe implementar a partir del interfaz OCI OracleOCIFailover().

Esta función callback es invocada durante el proceso de failover para notificar a la aplicación JDBC de cualquier evento generado y es responsabilidad de la aplicación implementar la respuesta necesaria en cada caso.

Finalmente, recordar que esta implementación de la interfaz OCI OracleOCIFailover() es opcional y para nada influye en el funcionamiento de general de la funcionalidad. La interfaz OracleOCIFailover() se define con los siguientes tipos y eventos:

```
public interface OracleOCIFailover{

        public static final int FO_SESSION = 1;
        public static final int FO_SELECT = 2;
        public static final int FO_NONE = 3;
        public static final int;

        public intcallbackFn (Connection conn, Object ctxt, int type, int event );
}
```

Fast Connection Failover (FCF)

Fast Connection Failover (FCF) permite que las aplicaciones JDBC aprovechen las ventajas de mecanismos de failover independientemente del tipo de driver JDBC, thin o thick. El único requisito es que solo está disponible a partir de la versión 10.1 de los drivers JDBC de Oracle.

Este caso, existe un gestor de conexiones o cache manager que limpia todas las conexiones inválidas de una aplicación JDBC después de un failover. En el momento que una aplicación JDBC intenta usar unas de esas conexiones invalidas, ésta recibe un error ORA-17008, "Closed Connection" y es la aplicación la que debe manejar la excepción y reconectarse.

FCF no reconecta las conexiones existentes, si el nodo que alberga la instancia cae, la aplicación debe cerrar las conexiones y adquirir nuevas conexiones a las instancias supervivientes del RAC. La forma de notificación que tiene FCF para comunicar a la aplicación JDBC que debe iniciar el proceso de reconexiones es a través de una excepción.

Estas excepciones están provocadas por diversos tipos de errores de tipo ORA-xxxxx, cada uno de ellos caracteriza a diferentes situaciones de caídas, cortes, etc. Por tanto la aplicación de prevenir todos estos tipos de ORA y reacción en consecuencia en función de las reglas de negocio.

Como soporte en este tipo de situaciones el API FCF ofrece una función para comprobar si el error producido es fatal o si por el contrario existe la posibilidad de continuar. Esta función se implementa a través del método isFatalConnectionError() de la clase OracleConnectionCacheManager.

En caso de que este método devuelva TRUE, se debe ejecutar el método getConnection sobre el datasource.

Para que una aplicación JDBC pueda hacer uso de la funcionalidad de FCF debe configurarse de la siguiente forma:

- Se debe configurar e iniciar ONS. Si ONS no está correctamente configurado, la creación de la cache de conexiones fallará levantando una excepción ONSException cuando se realice la primera petición getConnection().

- Además, se debe habilitar la propiedad FastConnectionFailoverEnabled antes de realizar la primera llamada al método getConnection de OracleDataSource, ya que cuando FCF está activo, se aplica a todas las conexiones del connection cache. Si la aplicación JDBC crea conexiones de forma explícita usando Connection Cache Manager, ésta debe habilitar el atributo setFastConnectionFailoverEnabled antes de adquirir la conexión.

- Por último, se debe usar SERVICE_NAME y no el SID de la base de datos propiedad URL del OracleDataSource.

Apéndices

Apéndice A : Eventos TAF

Los eventos TAF posibles son los siguientes:

- FO_SESSION, es equivalente al parámetro FAILOVER_MODE=SESSION de la sección CONNECT_DATA en la configuración tnsnames.ora. En este modo es necesario la re autentificación del usuario de sesión y que deben re ejecutarse todos los cursores abiertos por parte de la aplicación OCI.

- FO_SELECT, es equivalente al parámetro FAILOVER_MODE=SELECT de la sección CONNECT_DATA en la configuración tnsnames.ora. En este modo los cursores abiertos por una aplicación OCI pueden continuar con el fetching de las filas en caso de failover. Esto implica que la lógica del aplicativo es la encargada de mantener el estado del proceso de fetching de cada cursor abierto.

- FO_NONE, es equivalente al parámetro FAILOVER_MODE=NONE de la sección CONNECT_DATA en la configuración tnsnames.ora. Es el valor por defecto y es donde no existe funcionalidad de failover alguna. Esta función puede usarse para el caso de que no se desee realizar usar realizar ningún tipo de failover.

- FO_TYPE_UNKNOWN, implica que se ha especificado un tipo de failover erróneo retornado desde el driver OCI.

- FO_BEGIN, indica que se ha detectado la perdida de la conexión y que se ha iniciado el proceso de failover.

- FO_END, indica que el proceso de failover se ha completado satisfactoriamente.

- FO_ABORT, indica que el proceso de failover ha terminado no satisfactoriamente y que no hay opción de reintento.

- FO_REAUTH, indica que el usuario ha sido re autentificado.

- FO_ERROR, indica que el proceso de failover ha sido insatisfactorio de forma temporal, dando la oportunidad a la aplicación de manejar el error y volver a intentar el proceso de failover. Normalmente, el tratamiento de este evento se soluciona mediante un SLEEP y un reintento tras recibir el evento FO_RETRY.

- FO_RETRY, evento de finalización del escenario de FO_ERROR.

- FO_EVENT_UNKNOWN, evento del proceso de failover no esperado o erróneo.

Apéndice B : Ejemplos de conectividad a RAC desde Java.

Ejemplo Oracle WebLogic Server MultiDatasources

En este apartado se muestra un ejemplo de configuración que utiliza Oracle RAC con JDBC MDS de WLS.

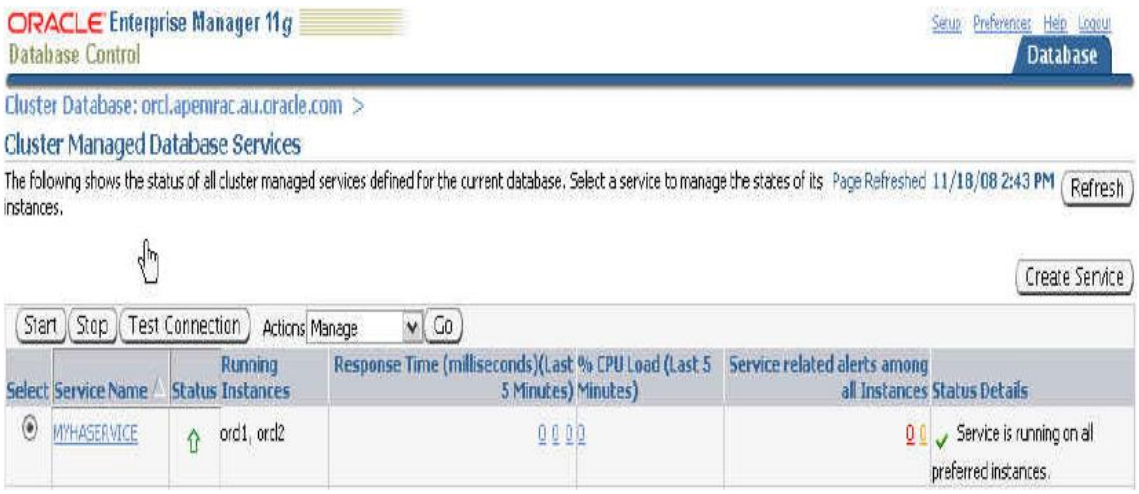
Configurando Oracle RAC y definiendo un Database Service

La primea tarea es configurar Oracle RAC correctamente para que pueda ser usado por tus aplicaciones middle-tier. Estos son los pasos:

- Logarse en Oracle Enterprise Manager Database Control como SYS.
- Hacer click en el link "Availability" en la parte superior de la página.
- Hacer click en el link "Cluster Managed Database Services".
- Suministrar las credenciales del clúster, así como las de la base de datos.
- Crear los servicios.
- Definir un servicio como se define en la tabla siguiente:

Parámetro	Valor
Service Name	MYHASERVICE
Enable Load Balancing Advisory	SELECTED
Throughput Radio Option	SELECTED
Connection Load Balancing Goal	LONG

Después de crear el servicio, se puede usar la página "Cluster manage Database Services" para verificar el estado del servicio como se muestra a continuación.

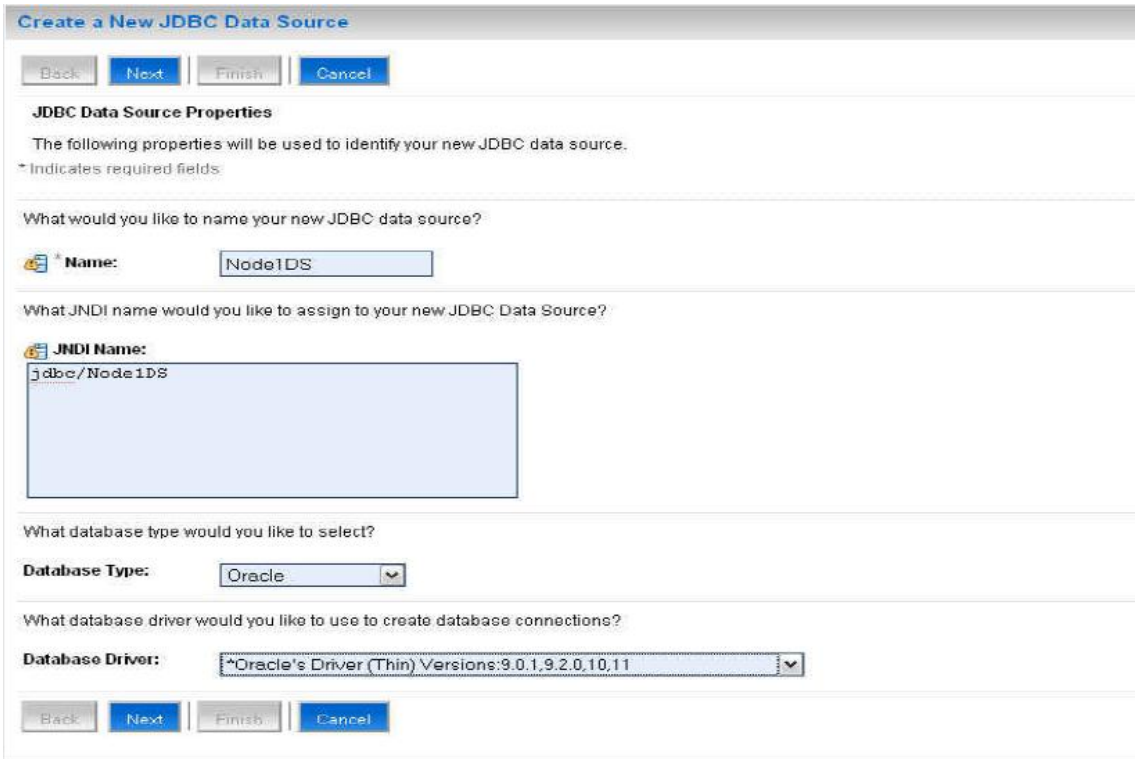


Configurando Multi Data Source en Oracle WebLogic Server

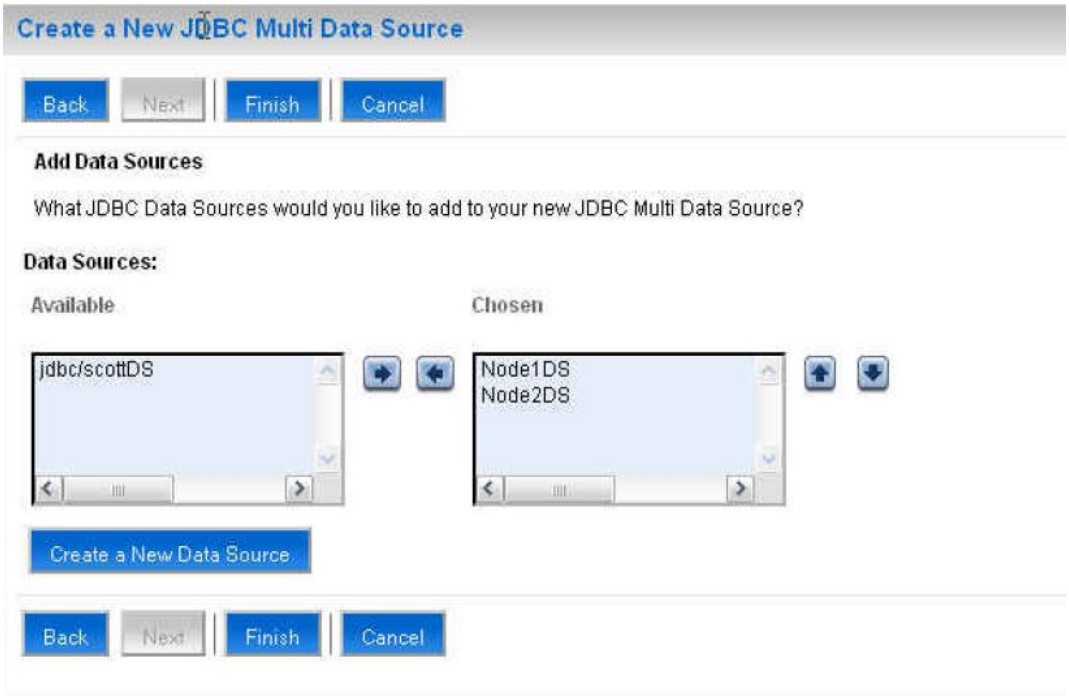
El siguiente paso de la configuración pasa por usar MDS de WLS. Como hemos visto anteriormente un MDS está compuesto de varios DS, cada uno de los cuales está configurado para conectarse a una instancia de Oracle RAC en el *backend*.

Las aplicaciones desplegadas deben obtener las conexiones a Oracle RAC usando el nombre JNDI específico del MDS.

Para ello, primero hay que crear un DS por cada nodo de Oracle RAC que se va a usar. A continuación se muestra como se haría con la consola de administración de WLS.



Después de la creación de los diferentes DS, hay que crear un MDS como se muestra a continuación:



El directorio del dominio, *config/jdbc*, deberá contener unos ficheros similares a los que se muestra a continuación:

```
[…/user_projects/domains/hadomain/config/jdbc]$ d
total 56
-rw-r--r-- 1  oracle oinstall 135  Nov 26 09:40 readme.txt
drwxr-xr-x 11 oracle oinstall 4096 Nov 27 12:31 ../
-rw-r--r-- 1  oracle oinstall 2259 Dec 3  07:48 jdbc2fscottDS-8614-jdbc.xml
drwxr-xr-x 2  oracle oinstall 4096 Dec 4  11:14 ./
-rw-r--r-- 1  oracle oinstall 857  Dec 4  11:15 jdbc2fracDS-4177-jdbc.xml
-rw-r--r-- 1  oracle oinstall 2508 Dec 4  12:23 Node1DS-0882-jdbc.xml
-rw-r--r-- 1  oracle oinstall 2508 Dec 4  12:27 Node2DS-9921-jdbc.xml
```

A continuación se muestra la configuración del MDS creado anteriormente:

```
<?xml version='1.0' encoding='UTF-8'?>
<jdbc-data-source
       xmlns="http://www.bea.com/ns/weblogic/jdbc-data-source"
       xmlns:sec="http://www.bea.com/ns/weblogic/90/security"
       xmlns:wls="http://www.bea.com/ns/weblogic/90/security/wls"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.bea.com/ns/weblogic/jdbc-datasource
       http://www.bea.com/ns/weblogic/jdbc-datasource/
       1.0/jdbc-data-source.xsd">
 <name>jdbc/racDS</name>
 <jdbc-connection-pool-params>
   <test-frequency-seconds>5</test-frequencyseconds>
 </jdbc-connection-pool-params>
 <jdbc-data-source-params>
    <jndi-name>jdbc/racDS</jndi-name>
    <algorithm-type>Load-Balancing</algorithm-type>
    <data-source-list>Node1DS,Node2DS</data-sourcelist>
    <failover-request-if-busy>false</failoverrequest-if-busy>
 </jdbc-data-source-params>
</jdbc-data-source>
```

Verificando las configuraciones

Para probar la configuración, existe una aplicación de ejemplo que tras ser desplegada, permite, en tiempo de ejecución, especificar la ubicación del MDS, el número de conexiones y el tiempo de espera entre peticiones de conexión.

La prueba se puede hacer para asegurar que el balanceo de carga entre los nodos de Oracle RAC se realiza correctamente, el MDS implementa la política elegida, ya sea load balancing o failover, y en general que la configuración es correcta para el escenario planteado.

A continuación se muestra un ejemplo de la pantalla de inicio de la citada aplicación.

Ejemplo JDBC Thin con Network failover o Connection failover

En este ejemplo se supone una base de datos llamada ORCL con dos instancias en dos host, rac1.es.oracle.com y rac2.es.oracle.com, con los TNSListener en los puertos 1521 en ambos.

Configuración TNS

```
ORCL =
  (DESCRIPTION =
    (ADDRESS_LIST =
        (ADDRESS =
          (PROTOCOL = TCP)
          (HOST = rac1.es.oracle.com)
          (PORT = 1521))
        (ADDRESS =
          (PROTOCOL = TCP)
          (HOST = rac2.es.oracle.com)
          (PORT = 1521)))
    (CONNECT_DATA =
      (SERVICE_NAME = ORCL))
   )
```

Ejemplo JDBC TestFailover.java

```
import java.sql.*;

class TestFailover
{
  public static void main (String args [ ]) throws SQLException
  {
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

    String url = "jdbc:oracle:thin:@DESCRIPTION = (ADDRESS_LIST = (ADDRESS =
         (PROTOCOL = TCP) (HOST = rac1.es.oracle.com) (PORT = 1521)) (ADDRESS =
         (PROTOCOL = TCP) (HOST = rac2.es.oracle.com) (PORT = 1521)))
         (CONNECT_DATA = (SERVICE_NAME = ORCL)))";
    Connection conn = DriverManager.getConnection (url, "system", "manager");

    Statement stmt = conn.createStatement ();
    ResultSet rset = stmt.executeQuery ("select host_name from v$instance");

    while (rset.next ())
         System.out.println (rset.getString (1));

    rset.close();
    stmt.close();
    conn.close();
  }
}
```

Ejemplo JDBC OCI8 con Transparent Application Failover

En este ejemplo se supone una base de datos llamada ORCL con dos instancias en dos host, rac1.es.oracle.com y rac2.es.oracle.com, con los TNSListener en los puertos 1521 en ambos.

Este ejemplo de TAF con JDBC OCI8 implementa una función OCI TAF Callback para operaciones DML y PL/SQL.

Configuración TNS

```
ORCL =
         (DESCRIPTION =
                  (LOAD_BALANCE= yes)
                  (FAILOVER=ON)
                  (ADDRESS =
                           (PROTOCOL = TCP)
                           (HOST = rac1.es.oracle.com)
                           (PORT = 1521))
                  (ADDRESS =
                           (PROTOCOL = TCP)
                           (HOST = rac2.es.oracle.com)
                           (PORT = 1521))
             (CONNECT_DATA =
```

```
        (SERVICE_NAME = ORCL)
        (FAILOVER_MODE=
                  (TYPE=SELECT)
                  (METHOD=BASIC))
    )
  )
```

Objetos de la base de datos necesarios

```
create table testme (c1 timestamp);

create or replace  procedure test_taf (p_date IN TIMESTAMP)
is
begin
        insert into testme values(p_date);
end;
/
```

Ejemplo JDBC DMLFailOver.java

```
import java.sql.*;
import java.net.*;
import java.io.*;
import java.util.*;
import oracle.jdbc.OracleConnection;
import oracle.jdbc.OracleOCIFailover;

public class DMLFailOver {
  static final String user = "scott";
  static final String password = "tiger";
  static final String driver_class = "oracle.jdbc.driver.OracleDriver";
  static final String URL = "jdbc:oracle:oci:@orcl";
  private static CallableStatement call;
  private static final String callString = "call test_taf (?)";
  private static Connection conn;
  private static boolean foFlag = false;

  public static void main (String[] args) throws Exception {
    System.out.println( "Begin ");
    CallBack   fcbk= new CallBack();
    Statement  stmt = null;
    ResultSet rset = null;
    // Load JDBC driver
    try {
      Class.forName(driver_class);
    }
    catch(Exception e) {
      System.out.println(e);
    }
    // Connect to the database
    conn = DriverManager.getConnection(URL, user, password);
    // register TAF callback function
    ((OracleConnection) conn).registerTAFCallback(fcbk, new Object());
          if ( conn != null ) {
       conn.setAutoCommit(false);
       DatabaseMetaData meta = conn.getMetaData ();

       System.out.println("\nDatabase      Product      Version      is           "      +
meta.getDatabaseProductVersion());
       System.out.println("\n==============\nJDBC    Driver    Name    is    ........    "    +
meta.getDriverName());
       System.out.println("\nJDBC Driver Version is ..... " + meta.getDriverVersion());
       System.out.println("\nJDBC URL " + meta.getURL());
       System.out.println("\n==============");
   }
    else {
      System.out.println( "FAILURE: Connection is NULL!" );
      System.out.print( "Exiting\n\n" );
      System.exit(0);
   }
    // Create a Statement
    stmt = conn.createStatement ();
    rset = stmt.executeQuery ("SELECT HOST_NAME, INSTANCE_NAME from v$instance");
      while (rset.next())
              {
              System.out.println ("Host name is "  + rset.getString(1));
              System.out.println ("Instance name is " +rset.getString(2));
              }
        stmt.close();
        rset.close();
    // Create a Statement
    call = conn.prepareCall( callString );
    for (int i=0; i<10; i++) {
      // Call pl/sql procedure
      doCall();
```

```
      // Sleep one second to make it possible to shutdown the DB.
      Thread.sleep(9000);
    } // End for
        // Close the connection
    conn.close();
    System.out.println( "Done ");
  } // End Main()

private static void doCall() throws SQLException {
    System.out.println( "about to set timestamp" );
    call.setTimestamp( 1, new Timestamp( System.currentTimeMillis() ) );
    System.out.println( "about to execute" );
    int code = 0;
    int rows = 0;
    foFlag = false;
    try
    {
      rows = call.executeUpdate();
    }
    catch (SQLException ex)
    {
    code = ex.getErrorCode();
    System.out.println("Error code in execute is :- " + code);

    if (code == 25408 && foFlag == true)
      {
        conn.rollback();
        call.close();
        call = conn.prepareCall(callString);
        System.out.println( "about to set timestamp" );
        call.setTimestamp( 1, new Timestamp( System.currentTimeMillis() ) );
        System.out.println( "about to execute" );
        rows = call.executeUpdate();
      }
      else
      {
        throw ex;
      }
    }
    System.out.println( "about to commit" );
    conn.commit();
    System.out.println( "commit finished" );
  } // End docall()
/* Define class CallBack */
public static class CallBack implements OracleOCIFailover {
   // TAF callback function
   public int callbackFn (Connection conn, Object ctxt, int type, int event) {

     String failover_type = null;

     switch (type) {
         case FO_SESSION:
                failover_type = "SESSION";
                break;
         case FO_SELECT:
                failover_type = "SELECT";
                break;
         default:
                failover_type = "NONE";
     }
     switch (event) {
       case FO_BEGIN:
            System.out.println(ctxt + ": "+ failover_type + " failing over...");
            break;
       case FO_END:
            System.out.println(ctxt + ": failover ended");
            boolean doReconnect = true;
            foFlag = true;
            break;
       case FO_ABORT:
            System.out.println(ctxt + ": failover aborted.");
            break;
       case FO_REAUTH:
            System.out.println(ctxt + ": failover.");
            break;
       case FO_ERROR:
            System.out.println(ctxt + ": failover error gotten. Sleeping...");
            // Sleep for a while
            try {
              Thread.sleep(100);
            }
            catch (InterruptedException e) {
               System.out.println("Thread.sleep has problem: " + e.toString());
            }
            return FO_RETRY;
       default:
```

```
                System.out.println(ctxt + ": bad failover event.");
                break;
            }
            return 0;
        }
} // End class Callback
} // End class jdemofo
```

Ejemplo JDBC Thin con Fast Connection Failover.

En este ejemplo se supone una base de datos llamada ORCL con dos instancias en dos host, rac1.es.oracle.com y rac2.es.oracle.com, con los TNSListener en los puertos 1521 en ambos.

Configuración TNS

```
ORCL =
        (DESCRIPTION =
                (LOAD_BALANCE= yes)
                (FAILOVER=ON)
                (ADDRESS =
                        (PROTOCOL = TCP)
                        (HOST = rac1.es.oracle.com)
                        (PORT = 1521))
                (ADDRESS =
                        (PROTOCOL = TCP)
                        (HOST = rac2.es.oracle.com)
                        (PORT = 1521))
            (CONNECT_DATA =
              (SERVICE_NAME = ORCL)
              (FAILOVER_MODE=
                        (TYPE=SELECT)
                        (METHOD=BASIC))
            )
          )
```

Ejemplo JDBC FCFDemo.java

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Properties;
import oracle.jdbc.pool.OracleDataSource;

public class FCFDemo {
      public staticvoid main(String[] args) throws InterruptedException {

               try {
                OracleDataSource ods = new OracleDataSource();
                ods.setUser("scott");
                ods.setPassword("tiger");
                String dbURL="jdbc:oracle:thin:@"
                        +"(DESCRIPTION=" +
                        "(ADDRESS=(PROTOCOL =TCP)" +
                        "(HOST=rac1.es.oracle.com)(PORT=1521))" +
                        "(ADDRESS=(PROTOCOL =TCP)" +
                        "(HOST=rac2.es.oracle.com)(PORT=1521))" +
                        "(LOAD_BALANCE=ON)(CONNECT_DATA=(SERVICE_NAME=ORCL)))";
                ods.setURL(dbURL);
                ods.setConnectionCachingEnabled(true);
                ods.setConnectionCacheName("cache");
                Properties prop = new Properties();
                prop.setProperty("MinLimit", "5");
                prop.setProperty("MaxLimit", "40");
                prop.setProperty("InitialLimit", "10");
                ods.setConnectionCacheProperties(prop);
                ods.setFastConnectionFailoverEnabled(true);

                Connection conn = ods.getConnection();
                System.out.println(conn);
                Statement stmt=null;
                ResultSetrs = null;

                try {
                 while (true){
                        rs =stmt.executeQuery("select instance_name from v$instance");
                        while(rs.next()) {
                                System.out.println("Instance name: " + rs.getString(1));
                        }
```

```
                    Thread.sleep(1000);
                   }
                  }
                  catch (SQLException sqle) {
                   conn =ods.getConnection(); //Re-get the conn
                   stmt =conn.createStatement();
                  }
                 }
                 catch(Exception e){
                  e.printStackTrace();
                 }
        }
}
```