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 Parallel Execution para Dataware Housing

Referencia documento: InfV5_JASAS_DWH_vs_PEX_v920.doc
Fecha: 16 de noviembre de 2018
Versión: 9.2.0

Registro de Cambios

Fecha	Autor	Versión	Notas
14 de Abril de 2011	Oracle ACS	2.2.0	Versión inicial
14 de Julio de 2011	Oracle ACS	2.3.0	Versión Julio 2011
13 de Octubre de 2011	Oracle ACS	2.4.0	Versión Octubre de 2011
14 de March de 2013	Oracle ACS	3.1.0	Versión Enero de 2012
14 de Marzo de 2013	Oracle ACS	4.1	Revisión de Marzo de 2013, contrato 2012-2014
13 de Junio de 2013	Oracle ACS	4.2	Revisión de Junio de 2013, contrato 2012-2014
17 de Octubre de 2013	Oracle ACS	4.3	Revisión de Octubre de 2013, contrato 2012-2014
16 de Julio de 2015	Enrique Ramiro	6.1	Revisión de Julio de 2015, contrato 2014-2016
16 de Diciembre de 2015	Enrique Ramiro	6.2	Revisión de Diciembre de 2015, contrato 2014-2016
16 de Junio de 2016	Enrique Ramiro	7.1	Revisión de Junio de 2016, contrato 2014-2016
16 de Noviembre de 2016	Enrique Ramiro	7.2	Revisión de Noviembre de 2016, contrato 2014-2016
16 de Junio de 2017	Enrique Ramiro	8.1	Revisión de Junio de 2017, contrato 2016-2018
16 de Noviembre de 2.017	Enrique Ramiro	8.2	Revisión de Noviembre de 2017, contrato 2016-2018
16 de Junio de 2.018	Enrique Ramiro	9.1	Revisión de Junio de 2018, contrato 2016-2018
16 de Noviembre de 2.018	Enrique Ramiro	9.2	Revisión de Noviembre de 2018, contrato 2016-2018

Revisiones

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

Distribución

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

Índice de Contenidos

Control de cambios

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

Introducción

Este documento recoge una serie de recomendaciones de Oracle Soporte planteadas como buenas prácticas de desarrollo para aplicaciones que hagan uso de Oracle RDBMS 12cR2 y Parallel Execution en entornos de Data Warehousing.

Estas recomendaciones están encaminadas a minimizar los posibles problemas de rendimiento en sistemas 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 12c, 11g y Parallel Execution para Data Warehousing 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 de tipo Data Warehousing a través de Oracle RDBMS 12cR2 y Parallel Execution.

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 la versión Oracle RDBMS 12cR2, aunque la mayoría de las recomendaciones son igualmente aplicables a versiones anteriores; hasta la 9iR2.

Best Practices de Parallel Executiong para Data Warehousing

Introducción

Hoy en día las bases de datos, independientemente de si son data warehouse, datastores operacionales o sistemas OLTP, contienen una gran cantidad de información. Sin embargo, encontrar y presentar información de la forma adecuada en el momento oportuno puede llegar a ser un reto debido a la gran cantidad de datos involucrados.

La ejecución paralela es la característica que se ocupa de este desafío. Utilizando paralelismo, pueden procesarse terabytes de datos en pocos minutos, no horas ni días. La ejecución en paralelo utiliza varios procesos para llevar a cabo una única tarea. Con este tipo de ejecución más eficaz, la base de datos puede aprovechar todos los recursos hardware – varios núcleos, varios canales de E/S, varios nodos en un clúster – y así procesar de una forma más eficientemente consultas y otras operaciones de base de datos.

Los grandes data warehouses suelen utilizar la ejecución en paralelo para conseguir un buen rendimiento. Las operaciones específicas de aplicaciones OLTP, tales como operaciones batch, también pueden beneficiarse significativamente de la ejecución en paralelo. El documento abarca tres temas fundamentales:

- Conceptos fundamentales de la ejecución paralela.
- Por qué se utiliza la ejecución en paralelo y cuáles son los principios fundamentales detrás de toda ejecución en paralelo.
- Aplicación de ejecución en paralelo y mejoras de Oracle.

Parallel Execution: Conceptos

La ejecución en paralelo, en adelante PEX, es un método comúnmente utilizado para acelerar las operaciones dividiendo las tareas en subtareas más pequeñas. En esta sección vamos a ver los principales argumentos de la ejecución en paralelo y los conceptos básicos.

¿Por qué usar PEX?

Imagine que su trabajo consiste en contar el número de coches en una calle. Hay dos maneras de hacer esto, una, puede ir por la calle solo y contar el número de coches o puede decírselo a un amigo y entonces comenzando cada uno por cada extremo de la calle, contar los coches hasta que se encuentren el uno con el otro. Luego para completar la tarea tendrán que sumar los resultados que cada uno ha obtenido.

Asumiendo que su amigo cuenta igual de rápido que lo hace usted, se completará la tarea de contar todos los coches en una calle en aproximadamente la mitad del tiempo en comparación a cuando el trabajo lo realizaba usted solo. Si este es el caso,

sus operaciones escalan linealmente; 2x el número de recursos, la mitad del tiempo total de proceso.

En una base de datos ocurre lo mismo que en el ejemplo de contar coches. Si se asigna el doble de los recursos y se logra un tiempo de procesamiento que es la mitad de lo que era con la cantidad original de recursos, entonces, la operación escala linealmente. Escalar linealmente es el objetivo final del procesamiento en paralelo, tanto de contar coches, como de entregar respuestas de una consulta de base de datos.

Algo de teoría sobre PEX

En el ejemplo de contar coches hemos hecho algunos supuestos básicos para obtener escalabilidad lineal. Estos supuestos reflejan parte de la teoría que hay detrás del procesamiento en paralelo.

En primer lugar se optó porque sólo uno de ustedes hiciera el recuento. En ese momento se decidió, como llamaríamos en una base de datos, el grado de paralelismo. En otras palabras, ¿cuántas personas serían necesarias para resolver el problema rápidamente en el caso ideal? Cuanto mayor es la carga de trabajo, más gente se podría utilizar y, por claro está, si se trata de una calle pequeña con 4 coches, debemos evitar cualquier paralelismo, ya que se necesitaría más tiempo en decidir quién empieza que en contar los coches.

Así que se decidió que merecía la pena tener dos personas coordinadas contando. En una base de datos esta elección se realiza mediante el optimizador de consultas en base al coste de la operación.

En segundo lugar, en el ejemplo de los coches se dividió el trabajo en dos partes iguales, se asumió que cada uno comenzó en un extremo opuesto de la calle y que contó a la misma velocidad que el otro. Lo mismo pasa en el procesamiento paralelo en una base de datos. El primer paso es dividir los datos en trozos de igual tamaño, que permitan su tratamiento en la misma cantidad de tiempo. Suele utilizarse algún tipo de algoritmo hash para dividir los datos.

Esta partición de datos se lleva a cabo de dos formas fundamentalmente. La diferencia principal es si la partición de datos física se utiliza como base - y por lo tanto existe un pre-requisito estático - para la paralelización del trabajo.

Estos enfoques fundamentales se conocen como arquitectura shared everything y arquitectura shared nothing, respectivamente.

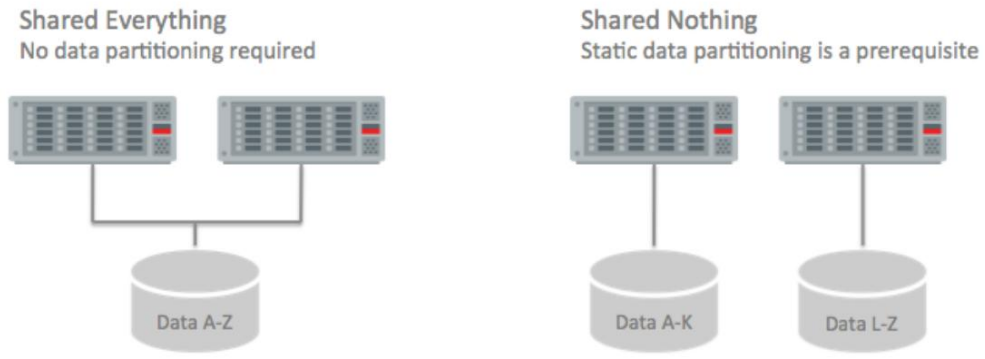


Figure 1: Shared everything versus shared nothing

En un sistema shared nothing, el sistema se divide en distintas unidades de procesamiento en paralelo. Cada unidad de proceso tiene su propio potencial de procesamiento (núcleos), su propio almacenamiento (disco) y sus núcleos de CPU son el único responsable de los datos almacenados en sus propios discos. La única manera de acceder a una parte específica de los datos es utilizar la unidad de procesamiento que posee este subconjunto de datos.

Estos sistemas son también comúnmente conocidos como sistemas de Procesamiento Paralelo Masivo (MPP). Para lograr una buena distribución de la carga de trabajo los sistemas shared nothing no usan un algoritmo hash para particionar los datos de forma homogénea entre todas las unidades de procesamiento disponibles. La estrategia de partición tiene que decidirse durante el inicio de la creación del sistema.

Como resultado, los sistemas shared nothing introducen, obligatoriamente, paralelismo fijo en sus sistemas con el fin de realizar operaciones que implican recorridos de tablas, el paralelismo fijo completo se basa en un particionamiento fijo y estático de los datos durante la creación de la base de datos u objeto.

La mayoría de los sistemas data warehouse que no son de Oracle son sistemas shared nothing. Por el contrario, Oracle Database se basa en una arquitectura shared everything. Esta arquitectura no exige partitioning predefinido de los datos para habilitar el paralelismo; pero utilizando Oracle Partitioning, Oracle Database puede ofrecer exactamente la misma capacidad de procesamiento paralelo que un sistema shared nothing. Lo que hace que desaparezcan las restricciones de acceso paralelo requeridos en la disposición de los datos. En consecuencia Oracle puede paralelizar casi todas las operaciones de varias maneras y grados, independientemente de la disposición de datos subyacente.

Al utilizar una arquitectura shared everything, Oracle permite ejecución flexible en paralelo y alta concurrencia sin sobrecargar el sistema, usando un superconjunto de capacidades de ejecución paralela a diferencia de los proveedores de shared nothing.

Usando la ejecución paralela o Parallel Execution (PEX)

Oracle Database proporciona funcionalidad para realizar tareas complejas en paralelo, sin intervención manual. Las operaciones que pueden ejecutarse en paralelo incluyen pero no están limitadas a:

- Cargas de datos
- Consultas
- Sentencias DML
- Backups RMAN
- Creación de objetos, como índices o tablas
- Recopilación de estadísticas
- Y más

Este documento se centra solamente en la ejecución paralela de SQL, que abarca consultas en paralelo, DML (Lenguaje de manipulación de Datos) paralelas y DDL (Lenguaje de Definición de Datos) paralelas. Aunque este documento se centra en Oracle Database 12cR2, la información contenida en este documento es genérica para Oracle Database desde 10gR2, a menos que se indique explícitamente.

Procesamiento de sentencias SQL paralelas

Cuando se ejecuta una sentencia SQL en Oracle Database, ésta se descompone en diferentes pasos u orígenes de filas, identificados con líneas diferentes en el plan de ejecución. A continuación puede verse un ejemplo en el que una sentencia SQL hace referencia a una sola tabla y su plan de ejecución. La sentencia devuelve el número total de clientes que existen en la tabla CUSTOMERS:

```
SELECT count(*) FROM customers c;
```

```
-----------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |     1 |     5   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE    |           |     1 |            |          |
|   2 |   TABLE ACCESS FULL| CUSTOMERS |   630 |     5   (0)| 00:00:01 |
-----------------------------------------------------------------------------
```

Un plan de ejecución más complejo podría ser uno que incluyera un join entre varias tablas. En el ejemplo siguiente, se solicita información acerca de las compras hechas por los clientes. Para esto, se necesita un join entre las tablas CUSTOMERS y SALES.

```
SELECT c.name, s.purchase_date, s.amount
FROM customers c, sales s
WHERE s.customer_id = c.id ;
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation            | Name      | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |   960 | 27840 |    11  (10)| 00:00:01 |       |       |
|*  1 |  HASH JOIN           |           |   960 | 27840 |    11  (10)| 00:00:01 |       |       |
|   2 |   TABLE ACCESS FULL  | CUSTOMERS |   630 | 12600 |     5   (0)| 00:00:01 |       |       |
|   3 |   PARTITION RANGE ALL|           |   960 |  8640 |     5   (0)| 00:00:01 |     1 |    16 |
|   4 |    TABLE ACCESS FULL | SALES     |   960 |  8640 |     5   (0)| 00:00:01 |     1 |    16 |
-------------------------------------------------------------------------------------------------
```

Si ejecuta una sentencia en paralelo, Oracle Database paralelizará tantos pasos diferentes como sea posible y reflejará esto en el plan de ejecución. Si tuviera que volver a ejecutar las dos sentencias anteriores en paralelo, podría obtener los siguientes planes de ejecución.

```
-----------------------------------------------------------------------------------------------
| Id  | Operation              | Name      | Rows  | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |           |     1 |     2   (0)| 00:00:01 |        |      |            |
|   1 |  SORT AGGREGATE        |           |     1 |            |          |        |      |            |
|   2 |   PX COORDINATOR       |           |       |            |          |        |      |            |
|   3 |    PX SEND QC (RANDOM) | :TQ10000  |     1 |            |          |  Q1,00 | P->S | QC (RAND)  |
|   4 |     SORT AGGREGATE     |           |     1 |            |          |  Q1,00 | PCWP |            |
|   5 |      PX BLOCK ITERATOR |           |   630 |     2   (0)| 00:00:01 |  Q1,00 | PCWC |            |
|   6 |       TABLE ACCESS FULL| CUSTOMERS |   630 |     2   (0)| 00:00:01 |  Q1,00 | PCWP |            |
-----------------------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name      | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |           |   960 | 27840 |     5  (20)| 00:00:01 |       |       |        |      |            |
|   1 |  PX COORDINATOR        |           |       |       |            |          |       |       |        |      |            |
|   2 |   PX SEND QC (RANDOM)  | :TQ10002  |   960 | 27840 |     5  (20)| 00:00:01 |       |       |  Q1,02 | P->S | QC (RAND)  |
|*  3 |    HASH JOIN BUFFERED  |           |   960 | 27840 |     5  (20)| 00:00:01 |       |       |  Q1,02 | PCWP |            |
|   4 |     PX RECEIVE         |           |   630 | 12600 |     2   (0)| 00:00:01 |       |       |  Q1,02 | PCWP |            |
|   5 |      PX SEND HASH      | :TQ10000  |   630 | 12600 |     2   (0)| 00:00:01 |       |       |  Q1,00 | P->P | HASH       |
|   6 |       PX BLOCK ITERATOR |           |   630 | 12600 |     2   (0)| 00:00:01 |       |       |  Q1,00 | PCWC |            |
|   7 |        TABLE ACCESS FULL| CUSTOMERS |   630 | 12600 |     2   (0)| 00:00:01 |       |       |  Q1,00 | PCWP |            |
|   8 |     PX RECEIVE         |           |   960 |  8640 |     2   (0)| 00:00:01 |       |       |  Q1,02 | PCWP |            |
|   9 |      PX SEND HASH      | :TQ10001  |   960 |  8640 |     2   (0)| 00:00:01 |       |       |  Q1,01 | P->P | HASH       |
|  10 |       PX BLOCK ITERATOR |           |   960 |  8640 |     2   (0)| 00:00:01 |     1 |    16 |  Q1,01 | PCWC |            |
|  11 |        TABLE ACCESS FULL| SALES     |   960 |  8640 |     2   (0)| 00:00:01 |     1 |    16 |  Q1,01 | PCWP |            |
-------------------------------------------------------------------------------------------------------------------
```

Estos planes parecen un poco más diferentes que el anterior, principalmente porque existen pasos de procesamiento logístico adicional durante el procesmiento paralelo que no aparecían antes.

La ejecución paralela SQL en Oracle Database se basa en unos pocos conceptos fundamentales. La siguiente sección trata estos conceptos que le ayudarán a comprender la configuración de la ejecución paralela en su base de datos así como la lectura de los planes de ejecución paralela SQL.

Queue Coordinator (QC) y procesos parallel execution (PX)

La ejecución paralela SQL en Oracle Database se basa en los principios fundamentales de un coordinador, conocido como Queue Coordinator, QC en adelante y ejecución paralela (PX) de procesos. El QC es el proceso que inicia la sentencia paralela SQL y los procesos PX son distintos procesos que realizan trabajo en paralelo. El QC reparte el trabajo entre los procesos PX y puede tener que realizar una parte mínima del trabajo – mayormente logística - que no puede ejecutarse en paralelo. Por ejemplo una consulta paralela con una operación SUM() necesita un sumatorio final de cada sumatorio individual realizado por cada proceso PX.

El QC puede identificarse fácilmente en los planes de ejecución paralela anteriores como 'PX COORDINATOR' (por ejemplo ID 1 en la Figura 3 mostrada arriba). El proceso que actúa como el QC de la operación paralela SQL es el propio proceso de la actual sesión de usuario.

Los procesos PX se toman de un pool de procesos PX disponibles globalmente y se asignan a una determinada operación. Su configuración se discute en una sección posterior). Todo el trabajo, mostrado después de la entrada del QC en los planes de ejecución, figura 3 y figura 4, es hecho por procesos PX.

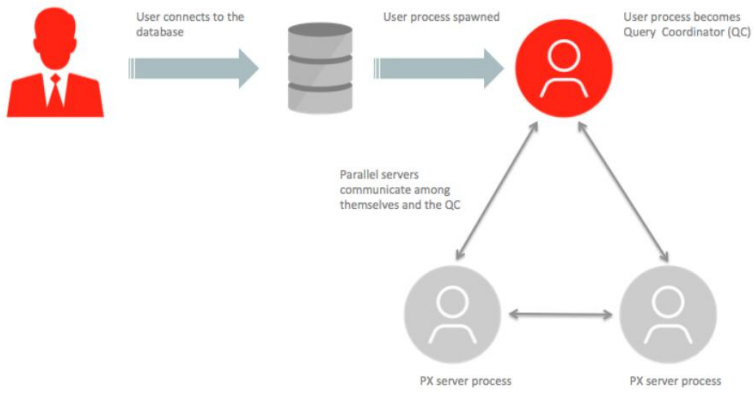


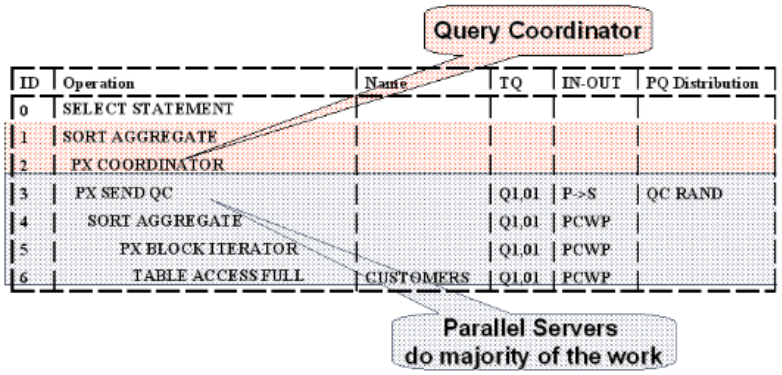
Figure : Parallel Execution with the Query Coordinator and a set of PX server processes

Los procesos PX pueden identificarse fácilmente por el sistema operativo, por ejemplo en Linux estos procesos son procesos oracle ORA_P***:

```
oracle   11191     1  0 Sep25 ?        00:10:26 ora_p024_dbm1
oracle   11195     1  0 Sep25 ?        00:11:04 ora_p025_dbm1
oracle   11197     1  0 Sep25 ?        00:11:12 ora_p026_dbm1
oracle   11199     1  0 Sep25 ?        00:10:47 ora_p027_dbm1
oracle   11201     1  0 Sep25 ?        00:10:56 ora_p028_dbm1
oracle   11203     1  0 Sep25 ?        00:11:00 ora_p029_dbm1
oracle   11205     1  0 Sep25 ?        00:11:07 ora_p030_dbm1
```

Volviendo al ejemplo de contar coches, habría una tercera persona – el QC- diciéndole a usted y a su amigo – dos procesos PX – que fueran de frente y contaran los coches; esto es equivalente a la operación con ID 2 en la figura 3, ilustrada en la figura 4.

Es exactamente lo mismo lo que se está haciendo en la carretera que lo que ocurre internamente en la base de datos con SQL y el plan de ejecución mostrado en la Figura 4: Usted y su amigo irán de frente y contarán los coches de su lado; esto es equivalente a las operaciones con ID4, ID5 y ID6, donde ID6 es el equivalente a decirles a ustedes que cuenten los coches de su lado de la carretera. Se detallará en la siguiente en la sección sobre granules).



Por último, cada uno de ustedes le dice a la tercera persona su subtotal individual (ID3) y entonces él hace el sumatorio del resultado final (ID1). Este es el traspaso de los procesos PX (procesos de hacen el trabajo efectivo) al QC para el "ensamblado" final del resultado que hay que devolver al proceso de usuario

Modelo Productor-Consumidor

Continuando con el ejemplo de contar coches, imagine que la tarea es ahora contar el número total de coches por color. Si usted y su amigo cubren cada uno un lado de la carretera, cada uno de ustedes podría ver los mismos colores y obtendría el subtotal para cada color, pero no el resultado para la calle completa.

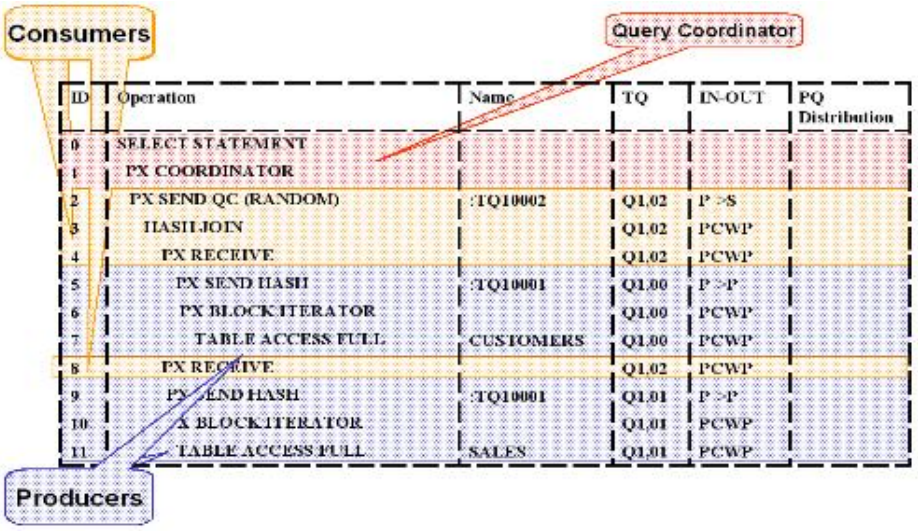
Usted podría ir de frente, memorizar toda esta información y al volver comunicársela a la tercera persona, "encargado". Pero entonces, este último tendrá que sumar todos los resultados el sólo – ¿qué pasa si todos los coches de la calle tienen color diferente? En este caso, la tercera persona tendría que deshacer el trabajo que usted y su amigo acaban de hacer.

Para paralelizar la cuenta, usted pregunta a dos amigos más para que le ayuden: Ambos caminan por el medio de la carretera, uno de ellos cuenta todos los colores oscuros y el otro los colores claros, suponiendo que esta "división de colores de coches" divide todos los colores por la mitad aproximadamente.

Cada vez que cuenta un coche nuevo, le comunica la cuenta a la persona encargada de ese color - usted produce la información, la distribuye en base a la información del color, y el contador del color consume la información. Al final, los dos amigos cuentan colores y le comunican su resultado al engargado; existen dos conjuntos, cada uno con dos amigos haciendo una parte del trabajo, trabajando mano a mano.

Así es cómo funciona la base de datos: para ejecutar una sentencia en paralelo eficientemente los conjuntos de procesos PX trabajan por pares: un conjunto producirá filas, productor, y otro conjunto consumirá las columnas, consumidor.

Por ejemplo, para el join paralelo entre las tablas SALES y CUSTOMERS, figura 2, las filas tienen que distribuirse en base a la clave del join para asegurar que las claves de ambas tablas se envían al mismo proceso PX que está haciendo el join. En este ejemplo un conjunto de procesos PX leen y envían los datos de la tabla CUSTOMERS, productor, y otro conjunto recibe los datos, consumidor, y hace el join con la tabla SALES.



Las operaciones que se procesan en el mismo conjunto de procesos PX pueden identificarse en el plan de ejecución mirando en la columna TQ. Como se muestra en la figura anterior, el primer conjunto esclavo (Q1,00) está leyendo la tabla CUSTOMERS en paralelo y produciendo filas que se envían al conjunto esclavo 2 (Q1,02) que consume esos registros y hace el join con el registro que procede de la tabla SALES (Q1,01).

Cuando los datos pasan de productores a consumidores puede verse una entrada de la forma: TQxxxxx (Table Queue x) en la columna NAME. Por ahora, es mejor obviar el contenido de las otras columnas.

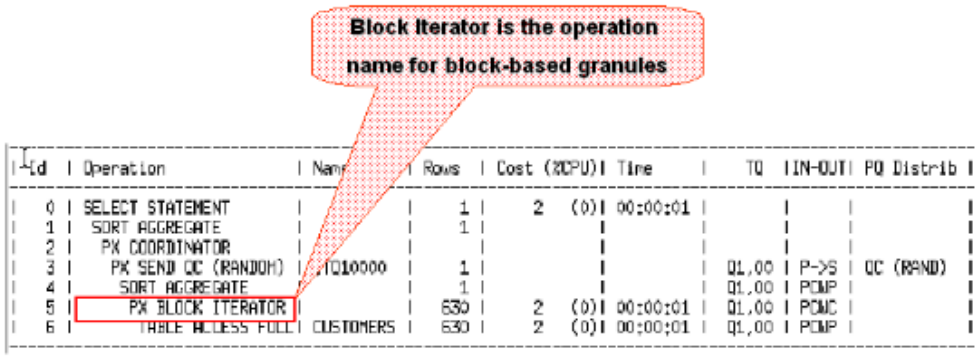
Esto tiene una importante consecuencia para el número de procesos PX que se intercambian en una operación paralela determinada: el modelo productor/consumidor espera dos conjuntos de procesos PX para una operación en paralelo, asi que el número de procesos paralelos es dos veces el grado de paralelismo (DOP) necesario. Por ejemplo, si el join paralelo en la figura anterior se ejecuta con un grado de paralelismo de 4, se usarán 8 procesos PX para esta sentencia, 4 productores y 4 consumidores.

El único caso en el que los procesos PX no trabajan por pares es si la sentencia es tan básica que un sólo conjunto de procesos PX puede completar la sentencia en paralelo. Por ejemplo select count(*) from customers necesita un único conjunto de procesos PX.

Granules o gránulos

Un granule o gránulo es la unidad de trabajo más pequeña en el acceso a datos. Oracle database usa una arquitectura shared everything, lo que desde una perspectiva de almacenamiento significa que cualquier núcleo de CPU en una configuración puede acceder a cualquier trozo de datos; esta es la diferencia fundamental referente a la arquitectura entre Oracle y otros proveedores de bases de datos del mercado. A diferencia de los demás sistemas, Oracle puede elegir una unidad de trabajo más pequeña dependiendo de los requisitos de la consulta.

El mecanismo básico que Oracle Database usa para distribuir el trabajo para la ejecución en paralelo es crear rangos de bloques en el disco, conocidos como gránulos basados en bloque. Esta metodología es única de Oracle y es independiente de si los objetos subyacentes han sido particionados. El acceso a los objetos subyacentes se divide en un número grande de gránulos, que se entregan a procesos PX para trabajar en ellos (y cuando un proceso PX termina el trabajo para un gránulo se le da el siguiente).



Id	Operation	Name	Rows	Cost (%CPU)	Time	TQ	IN-OUT	PQ Distrib
0	SELECT STATEMENT		1	2 (0)	00:00:01			
1	SORT AGGREGATE		1					
2	PX COORDINATOR							
3	PX SEND QC (RANDOM)	:TQ10000	1			Q1,00	P->S	QC (RAND)
4	SORT AGGREGATE		1			Q1,00	PCMP	
5	PX BLOCK ITERATOR		630	2 (0)	00:00:01	Q1,00	PCMC	
6	TABLE ACCESS FULL	CUSTOMERS	630	2 (0)	00:00:01	Q1,00	PCMP	

El número de gránulos es siempre mucho mayor que el DOP necesario para obtener una distribución uniforme del trabajo entre los procesos de ejecución paralela. La operación 'PX BLOCK ITERATOR', mostrada en la figura anterior, es la iteración sobre todos los gránulos de rango de bloques.

Aunque los gránulos basados en bloques son el fundamento de la ejecución paralela para la mayoría de las operaciones, hay algunas operaciones que pueden beneficiarse de la estructura de datos subyacente y aprovechar las particiones individuales como gránulos de trabajo. Con particiones basadas en gránulos sólo un proceso PX realiza el trabajo para todos los datos de una partición. El Oracle Optimizer considerará gránulos basados en particiones si el número de (sub)particiones accedidas en la operación es al menos igual al DOP, e idealmente mucho más alto si puede haber sesgo en los tamaños de las diferentes (sub)particiones.

Las operaciones más comunes que usan gránulos basados en particiones son partition-wise joins, como se verá más adelante.

En base a la sentencia SQL y el grado de paralelismo, Oracle Database dedice si se conseguirá una ejecución más óptima con gránulos basados en bloques o con gránulos basados en particiones; no se puede influir en este comportamiento.

En el ejemplo de contar coches, un lado de la calle – o incluso un bloque de una calle larga- podría considerarse equivalente a un gránulo basado en bloque. El volumen de datos existente – la calle – se subdivide en trozos físicos en los que los procesos paralelos – usted y su amigo – trabajan independientemente.

Redistribución de los datos

Las operaciones paralelas, excepto las más básicas, generalmente necesitan redistribición de datos. La redistribución de datos se necesita para realizar operaciones como ordenaciones paralelas, agregaciones o joins. A nivel de gránulo de bloques no existe conocimiento sobre los datos contenidos en un gránulo determinado. Los datos tienen que redistribuirse antes de que una operación posterior utilice el contenido actual ¿Recuerda el último ejemplo de coches? El color del coche importa, pero usted no sabe – o no controla – qué colores de coches están aparcados en la calle. Usted redistribuyó la información referente a la cantidad de coches por color a sus dos amigos adicionales, distribuyendo por colores, permitiéndoles hacer la cuenta total de los colores de los que están a cargo.

La redistribución de datos se lleva a cabo entre los distintos procesos PX ya sea dentro de una sola máquina, o, entre varias máquinas en un Real Application Clusters (RAC). Por supuesto, en este último caso, la comunicación de la interconnect se utiliza para la redistribución de datos.

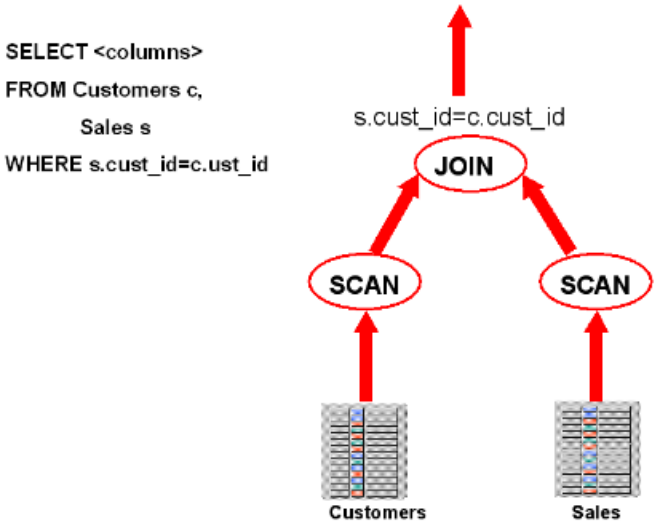
La redistribución de datos no es única de Oracle database. De hecho, este es uno de los principios fundamentales del procesamiento paralelo, y se usa en muchos productos que proporcionan capacidades paralelas. La diferencia fundamental y ventaja de las capacidades de Oracle, sin embargo, es que el acceso paralelo (visto en los gránulos de la sección anterior) y por lo tanto la redistribución necesaria de los datos no está limitada a determinadas arquitecturas hardware o configuraciones de bases de datos.

Los sistemas de base de datos Shared nothing también exigen la redistribución de datos a menos que las operaciones se puedan aprovechar partition-wise joins. En sistemas shared-nothing las operaciones paralelas que no pueden beneficiarse de partition-wise joins, tales como un join sobre tres tablas con dos claves diferentes, suelen hacer un uso excesivo de la comunicación de la interconnect. Dado que la base de datos Oracle también permite la ejecución en paralelo en un solo nodo, las operaciones paralelas no siempre tienen que utilizar la comunicación de la interconnect, evitando así un posible cuello de botella en la interconnect.

La siguiente sección explicará las capacidades de redistribución de datos de Oracle usando un simple ejemplo de joins de tablas sin estructura de datos secundaria, tales como índices o vistas.

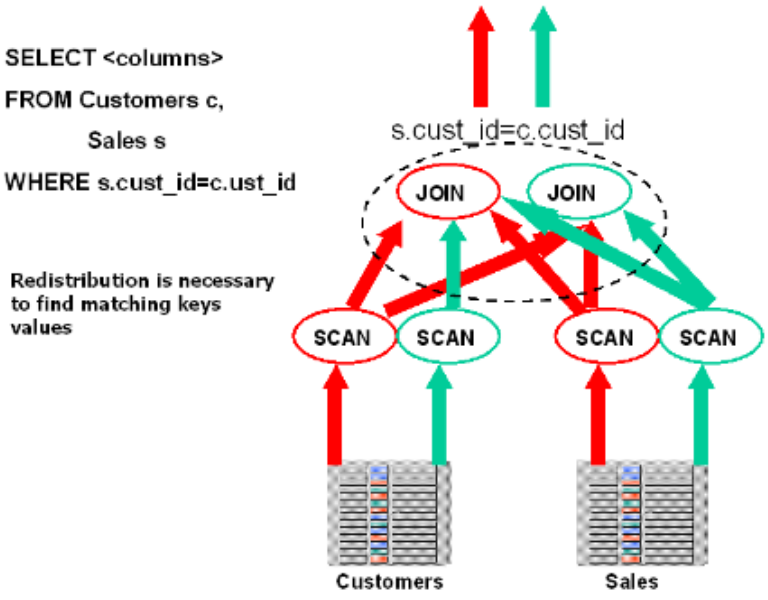
Join Secuencial

En un join secuencial o *serial join*, una sola sesión lee de ambas tablas y realiza el join. En este ejemplo se supone que las tablas CUSTOMERS y SALES están implicadas en el join. La base de datos utiliza full table scans para acceder a ambas tablas. Para un join secuencial una única sesión secuencial, puede realizar el join completo porque todos los valores de la tabla CUSTOMERS que concuerdan se leen por un solo proceso. La figura siguiente muestra el join secuencial.



Joins Paralelos

Al procesar el mismo join en paralelo, será necesaria la redistribución de filas. Los procesos PX recorren partes de cada tabla basándose en rangos de bloques y para finalizar el join, las filas se redistribuyen entre los procesos PX. La siguiente figura muestra la redistribución de datos para un join paralelo con un DOP de 2, representados por flechas de diferentes tonalidades respectvamente. Ambas tablas se leen en paralelo por los procesos A y B, usando gránulos en rango de bloques y a continuación, cada proceso PX tiene que redistribuir su conjunto de resultados en base a la clave del join para el siguiente operador del join paralelo.



Hay muchos métodos de redistribución de datos. Los siguientes 5 son los más comunes:

- HASH: La redistribución hash es muy común en la ejecución en paralelo para lograr una distribución equitativa del trabajo entre procesos PX, en base a una distribución hash. La (re)distribución hash es el mecanismo básico de ejecución paralela para la mayoría de sistemas de base de datos data warehouse.

- BROADCAST: La redistribución broadcast se da cuando uno de dos conjuntos de resultados en una operación join es mucho más pequeño que el otro conjunto de resultados. En lugar de redistribuir las filas de los dos conjuntos de resultados, la base de datos envía el conjunto de resultados más pequeño a todos los procesos PX para garantizar que los procesadores individuales son capaces de terminar sus operaciones join. El resultado pequeño puede ejecutarse de forma secuencial o en paralelo.

- RANGE: La redistribución por rango suele usarse normalmente para operaciones de ordenamiento. Los distintos procesos PX trabajan sobre rangos de datos de forma que el QC no tiene que hacer ningún ordenamiento solo tiene que presentar los resultados de los distintos procesos paralelos en el orden correcto.

- KEY: La redistribución por clave asegura que los conjuntos de resultados con distintos valores de clave se agrupan. Esta es una optimización que suele usarse principalmente en partial partition-wise joins (ver más adelante) para garantizar que se redistribuye sólo una parte del join.

- HYBRID HASH: La redistribución hash híbrida, introducida en Oracle Database 12c, es una técnica de distribución adaptativa (*adaptive*) que retrasa la decisión del método de distribución final hasta el tiempo de ejecución y se basa en el tamaño del conjunto de resultados. Un nuevo paso del plan llamado STATISTICS COLLECTOR se coloca frente a la nueva distribución híbrida de hash, cuenta el número de filas devueltas desde los procesos PX y comprueba el recuento contra un umbral máximo. Si se supera el umbral, es más efectivo en cuanto a coste distribuir las filas según una distribución hash. Si el tamaño del conjunto de resultados completo es igual o menor al umbral, entonces los datos se distribuyen por método broadcast.

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop	TQ	IN-OUT	PQ Distrib
0	SELECT STATEMENT		960	27840	5 (20)	00:00:01					
1	PX COORDINATOR										
2	PX SEND QC (RANDOM)	:TQ10002	960	27840	5 (20)	00:00:01			Q1,02	P->S	QC (RAND)
* 3	HASH JOIN BUFFERED		960	27840	5 (20)	00:00:01			Q1,02	PCWP	
4	PX RECEIVE		630	12600	2 (0)	00:00:01			Q1,02	PCWP	
5	PX SEND HYBRID HASH	:TQ10000	630	12600	2 (0)	00:00:01			Q1,00	P->P	HYBRID HASH
6	STATISTICS COLLECTOR								Q1,00	PCWC	
7	PX BLOCK ITERATOR		630	12600	2 (0)	00:00:01			Q1,00	PCWC	
8	TABLE ACCESS FULL	CUSTOMERS	630	12600	2 (0)	00:00:01			Q1,00	PCWP	
9	PX RECEIVE		960	8640	2 (0)	00:00:01			Q1,02	PCWP	
10	PX SEND HYBRID HASH	:TQ10001	960	8640	2 (0)	00:00:01			Q1,01	P->P	HYBRID HASH
11	PX BLOCK ITERATOR		960	8640	2 (0)	00:00:01	1	16	Q1,01	PCWC	
12	TABLE ACCESS FULL	SALES	960	8640	2 (0)	00:00:01	1	16	Q1,01	PCWP	

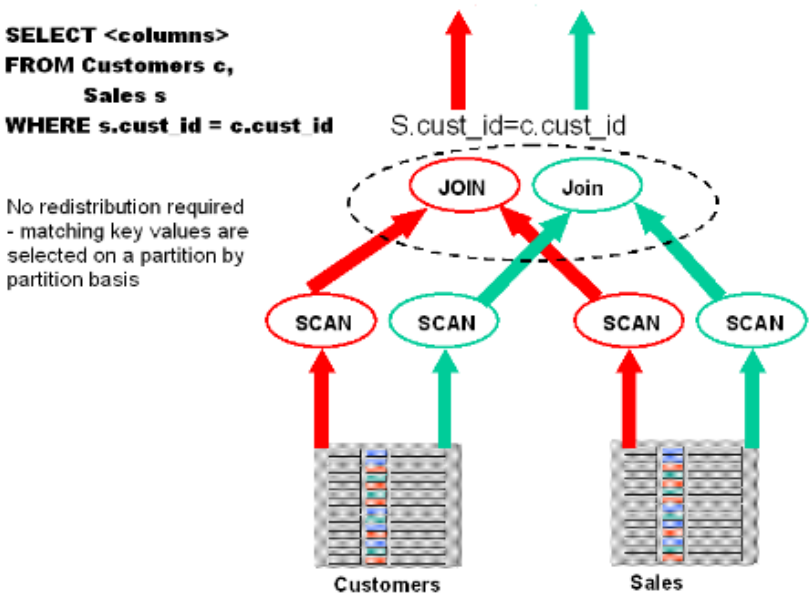
Figure : Execution plan for hybrid hash distribution

En este plan de ejecución podemos ver el método de redistribuciónen las líneas 5 y 10, la columna 'PQ Distrib' muestra HYBRID HASH y la columna 'Operation' muestra PX SEND HYBRID HASH. El nuevo paso del plan STATISTICS COLLECTOR está en la línea 6.

Como una variación en los métodos de distribución de datos, puede ver el sufijo LOCAL en el plan de ejecución paralela en una base de datos RAC. La resistribución LOCAL es una optimización de RAC para minimizar el tráfico de interconnect en consultas paralelas entre nodos. Por ejemplo, puede ver una redistribución BROADCAST LOCAL en un plan de ejecución indicando que el conjunto de filas se produce en el nodo local y solo se envían a los procesos PX en ese nodo.

Joins paralelos de tipo partition-wise

Si al menos una de las tablas accedidas en el join ha sido particionada por la clave del join, la base de datos puede decidir si usar partition-wise join. Si ambas tablas están igualmente particionadas por la clave de join, la base de datos puede usar un full partition wise join. En otro caso, podría usarse un partial partition-wise join en una de las tablas, particionadas dinámicamente en memoria, seguido de un full partition-wise join.



Un partition-wise join no necesita ninguna redistribución de datos porque un solo proceso PX trabajará en las particiones equivalentes de ambas tablas del join.

Como se muestra en la figura anterior, el proceso PX A lee datos de la partición de la tabla CUSTOMERS y datos de la partición de la tabla SALES; el equi-partitioning en ambas tablas por la clave del join garantiza que no habrá filas coincidentes en el join que se salgan fuera de esas dos particiones. El proceso PX siempre será capaz de terminar el join completo leyendo solo las particiones que tengan coincidencias. Lo mismo ocurre con el proceso PX B y para cualquier par de particiones en las dos tablas. Debe tenerse en cuenta que los partition-wise joins usan gránulos basados en particiones en lugar de gránulos basados en bloques.

Los partition-wise joins hablitan el funcionamiento de los sistemas shared nothing. Los sistemas shared nothing normalmente escalan bien siempre y cuando aprovechen los partition-wise joins. Como resultado, la elección de partitioning, distribución, en un sistema shared nothing es la clave, así como la ruta a las tablas. Las operaciones que no usn partition-wise joins en un sistema MPP no suelen escalar de forma óptima.

Ejecución Paralela en Memoria

A diferencia del procesamiento paralelo tradicional donde los datos pasan por alto cualquier caché compartida y se transfieren directamente a la memoria (PGA) de los procesos PX, la ejecución paralela en memoria (PX en memoria ó *in-memory PX*)

aprovecha la caché de memoria compartida (SGA) para almacenar datos para el posterior procesamiento paralelo. In-Memory PX aprovecha la memoria cada vez mayor de los servidores de bases de datos de hoy en día; esto es especialmente beneficioso en entornos de clúster a gran escala donde la cantidad agregada total de memoria puede ser múltiplos de Terabytes incluso cuando un servidor de base de datos individual "solo" contiene decenas o cientos de Gigabytes de memoria. Con in-memory PX, Oracle utiliza la caché de memoria agregada de los servidores en un entorno de Real Application Clusters (RAC) para cachear determinísticamente los objetos distribuidos en la memoria de todos los nodos. Esto garantiza que las consultas paralelas posteriores lean datos de la memoria caché de todos los nodos que pueden acelerar enormemente los tiempos de procesamiento en lugar de leer datos del almacenamiento.

Oracle Database 12c Release 12.1.0.2 ha introducido la opción Oracle Database In-Memory, un almacén de datos en memoria columnar diseñado específicamente para el procesamiento en memoria para permitir el análisis en tiempo real de grandes volúmenes de datos. Su formato columnar comprimido y los algoritmos de procesamiento de datos optimizados para el in-memory proporcionan el procesamiento de la memoria más óptimo posible. Aprovechar la nueva tecnología In-Memory de Oracle es el método recomendado para el procesamiento en memoria. Para obtener información detallada acerca de esta función, consulte el white paper "Oracle Database In-Memory".

Los sistemas sin la opción de base de datos en memoria también pueden aprovechar el PX en memoria, pero no podrán usar el almacenamiento en columna comprimido en memoria y los algoritmos optimizados en memoria. A partir de Oracle Database 11g Release 2, la base de datos está utilizando la buffer caché estándar de base de datos para PX en memoria, la misma caché que se utiliza para el procesamiento transaccional en línea. Dado que utilizan la misma caché, existe cierto riesgo de "competencia" para la buffer caché entre OLTP y operaciones paralelas, para garantizar que las operaciones paralelas no ocupen toda la caché, la Base de datos Oracle limita el porcentaje de la bufer caché que puede usar in-memory PX al 80%. Dependiendo de los volúmenes de datos procesados en paralelo y del requerimiento de memoria de las aplicaciones OLTP, los beneficios de la PX en memoria se pueden limitar con este modelo. Lo más común es que los sistemas con volúmenes de datos más pequeños y con características de carga de trabajo más mixtas (y cambiantes) se beneficien de este enfoque.

Para ampliar la aplicabilidad de PX en memoria, tanto en términos del tamaño de los objetos elegibles para el procesamiento en memoria como para proporcionar un optimizado full-table-scan-aware caching, Oracle Database 12c ha introducido la caché automática de tabla grande ó *Automatic Big Table Caching* (ABTC) que se reserva una parte dedicada de la memoria buffer caché específicamente para el procesamiento paralelo en memoria.

Con ABTC, una parte de la memoria buffer cache de la base de datos se reserva para almacenar objetos grandes (o partes de ellos) en la memoria para que más consultas puedan aprovechar la ejecución paralela en memoria. El tamaño de los objetos que pueden almacenarse en caché puede ser hasta tres veces mayor que la memoria reservada disponible. ABTC utiliza un segmento optimizado y un algoritmo temperature-based para garantizar el uso óptimo de la memoria caché disponible.

La configuración de los dos parámetros siguientes es necesaria para habilitar el in-memory PX con ABTC:

- PARALLEL_DEGREE_POLICY: tiene que establecerse en AUTO o ADAPTIVE

- DB_BIG_TABLE_CACHE_PERCENT_TARGET: especifica el porcentaje de la memoria buffer caché de la base de datos que se reservará para in-memory PX.

Con in-memory PX y ABTC, la base de datos decide si los objetos a los que accede la sentencia deben almacenarse en caché en ABTC o no. Un objeto puede ser una tabla, un índice o, en el caso de los objetos particionados, una o más particiones.

Esta decisión se basa en un conjunto avanzado de heurísticas que incluyen el tamaño de un objeto, la frecuencia a la que se accede al objeto y el tamaño de ABTC. Si el objeto cumple estos criterios, el procesamiento en memoria se habilitará y el objeto al que se accedió se almacenará en caché en ABTC. En el caso de un entorno de RAC con múltiples nodos, el objeto se fragmentará (se dividirá en partes) y se distribuirá a todos los nodos participantes; cada fragmento se asignará de forma determinista (por afinidad) a un nodo de RAC específico y se almacenará en su caché ABTC. Los fragmentos pueden ser rangos físicos de bloques de una tabla o en el caso de particiones individuales de objetos particionados.

Una vez que se ha mapeado un fragmento, todos los accesos subsiguientes de ese fragmento ocurrirán en ese nodo. Si una subsiguiente sentencia SQL paralela requiere los mismos datos desde cualquier nodo en el clúster, los procesos PX en los nodos donde residen los datos accederán a los datos en su caché ABTC y devolverán solo el resultado al nodo donde estaba la instrucción emitida; no se mueven datos entre los nodos a través de la Cache Fusion.

Si un objeto no se considera almacenado en caché, se accederá a través de direct path IO.

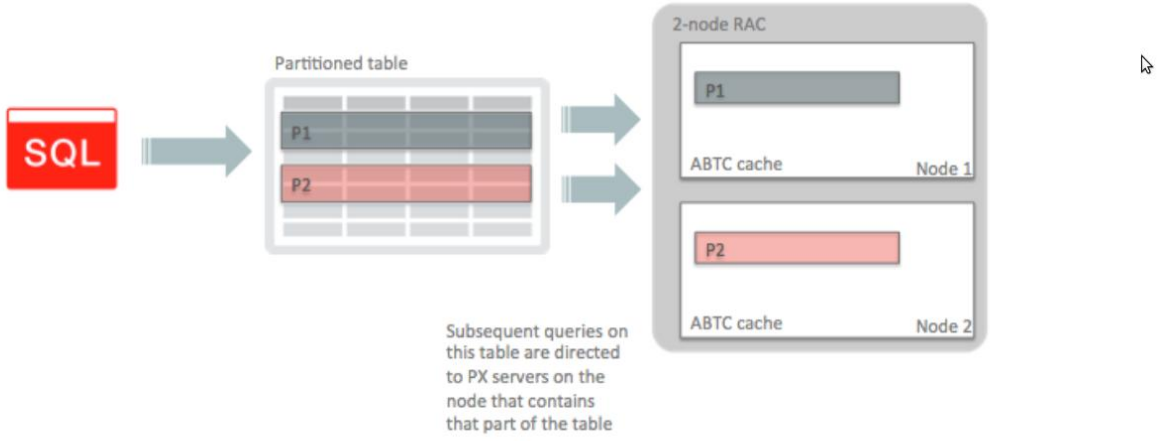


Figure: Sample data distribution for in-memory parallel execution for a partitioned table across a two node RAC cluster

Programación paralela para funciones de tablas

Aunque no es un concepto nuevo en Oracle Database 12cR2, las funciones de tabla son un aspecto importante en el paralelismo. El objetivo del conjunto de funciones de tabla es construir un pipeline de procesamiento paralelo aprovechando el marco de procesamiento de la base de datos.

Una function de tabla encapsula lógica compleja en construcciones PL/SQL, a la vez que permite procesar datos en paralelo. Una función de tabla también permite pasar los datos al siguiente consumidor, construyendo el pipeline con productores y consumidores. Este comportamiento puede verse mejor observando un pequeño ejemplo en SQL:

```
select *
from table(oracle_map_reduce.reducer(cursor(
      select * from table(oracle_map_reduce.mapper(cursor(
            select * from sls))) map_result)));
```

Como puede verse, la acción más interna es un select de una tabla, a continuación estos datos se transmiten a una función de tabla paralela que después de manipular los datos los transmite a la siguiente función de tabla. Esa función de tabla actúa en este caso como origen de los datos para el último select.

Control de la ejecución paralela

Para usar la ejecución en paralelo de una manera eficiente, debe considerar cómo habilitarlo, cómo especificar el grado de paralelismo ó Degree Of Parallelism (DOP) para las instrucciones y cómo usarlo en entornos concurrentes donde muchas sentencias paralelas pueden estar corriendo al mismo tiempo.

Habilitando la ejecución en paralelo:

De forma predeterminada, la base de datos Oracle está habilitada para la ejecución paralela de consultas y sentencias DDL. Para sentencias DML, debe habilitarlo a nivel de sesión con una instrucción ALTER SESSION:

```
ALTER SESSION ENABLE PARALLEL DML;
```

Por defecto, Oracle Database está habilitaba para la ejecución paralela y cuando está activo Automatic Degree of Parallelism, Auto DOP, la base de datos automáticamente decide si una sentencia debe ejecutarse en paralelo o no y el DOP que debe usarse. La decisión de usar ejecución paralela y la elección del DOP se basa en los recursos que necesita una sentencia.

Si elige no activar Auto DOP, tendrá tres formas de habilitar la ejecución paralela de una consulta:

1) Habilitar la ejecución paralela en la tabla:

```
alter table sales parallel ;
alter table customers parallel ;
```

Use este método si normalmente quiere ejecutar operaciones de acceso sobre tablas en paralelo.

2) Usar la palabra parallel en la sentencia SQL:

```
SELECT /*+ parallel(c) parallel(s) */ c.state_province,
sum(s.amount) revenue
FROM customers c, sales s
WHERE s.cust_id = c.cust_id
AND s.purchase_date
```

```
BETWEEN TO_DATE('01-JAN-2007','DD-MON-YYYY')
AND TO_DATE('31-DEC-2007','DD-MON-YYYY')
AND c.country = 'United States'
GROUP BY c.state_province;
```

Este método es útil principalmente para pruebas, o si tiene una sentencia en particular o un conjunto de sentencias que quiere ejecutar en paralelo, pero la mayoría de las sentencias se ejecutan secuencialmente.

3) Usar el commando alter session:

```
alter session force parallel query ;
```

Ahora que sabe cómo habilitar la ejecución paralela, puede preguntarse cómo controlar el número de procesos PX usados por cada sentencia SQL. Obviamente, si quiere usar muchos recursos para reducir los tiempos de respuesta, pero demasiadas operaciones hacen esta aproximación, el sistema puede quedarse sin recursos pronto, y por tanto no podrá usar más recursos de los que tiene.

Oracle Database también establece límites y ajustes para prevenir la sobrecarga del sistema y asegurar que la base de datos se mantiene disponible para las aplicaciones. El parámetro de inicialización de la base de datos PARALLEL_MAX_SERVERS es un buen ejemplo de uno de estos límites. Todos los procesos en la base de datos necesitan recursos, incluyendo memoria, CPU y recursos E/S. El sistema no asignará más procesos PX a un sistema de lo que indique la configuración del parámetro de inicialización.

Análisis de su carga de trabajo

La ejecución en paralelo puede habilitar una única operación que utiliza todos los recursos del sistema. Aunque puede no ser un problema en ciertos escenarios hay muchos casos en los que esto no es deseable.

Tenga en cuenta la carga de trabajo que quiere aplicar a la ejecución paralela para obtener un uso óptimo del sistema a la vez que se satisfacen los requisitos del sistema.

Carga de trabajo monousuario

Con carga de trabajo monousuario en la base de datos se ejecuta una única operación y el objetivo de esta operación es acabar tan rápido como sea posible. Un ejemplo de este tipo de carga de trabajo es una carga por lotes de gran tamaño que se realiza durante la noche que involucra grandes tablas de la base de datos y grandes recopilaciones de estadísticas. Con este tipo de carga todos los recursos pueden asignarse para mejorar el rendimiento de una operación.

Carga de trabajo multiusuario concurrente

La mayoría de los entornos en producción tienen carga de trabajo multiusuario. Los usuarios que ejecutan consultas concurrentemente, como consultas tipo ad-hoc y/o operaciones concurrentes de carga de datos. En un entorno multiusuario, los recursos de la carga de trabajo deben dividirse entre las operaciones concurrentes. Los usuarios finales esperan que una buena cantidad de recursos sean asignados a sus operaciones y así obtener tiempos de respuesta predecibles.

Administrar el grado de paralelismo

El número de procesos paralelos asociados a una operación se conoce como grado de paralelismo (DOP). El framework de ejecución paralela de Oracle, le permite elegir de forma explícita - o incluso forzar – un DOP determinado, o puede confiar en Oracle para controlarlo. El uso de Auto DOP es la forma recomendada por Oracle para controlar la ejecución en paralelo en Oracle Database 12c.

Paralelismo por defecto

En el ejemplo anterior se especificaba como un objeto era accedido en paralelo. No se especificó el DOP. En estos casos Oracle utiliza el llamado paralelismo DEFAULT. El paralelismo DEFAULT usa una fórmula que determina el DOP basándose en la configuración del sistema y parámetros de inicialización:
- PARALLEL_THREADS_PER_CPU * CPU_COUNT en una BBDD single-instance y
- PARALLEL_THREADS_PER_CPU * SUM(CPU_COUNT) en entornos RAC

Así que en un clúster de cuatro nodos donde cada nodo tiene 8 núcleos de CPU y un PARALLEL_THREADS_PER_CPU de 2; el DOP por defecto sería 2 * 8 * 4 = 64.

Se puede usar una de las siguientes vías para utilizar un paralelismo DEFAULT:

1. Usar la cláusula "parallel" a nivel de objeto:
```
ALTER TABLE customers PARALLEL;
```

2. Usar el hint "parallel(default)" a nivel de sentencia:
```
SELECT /*+ parallel(default) */ COUNT(*) FROM customers;
```

3. Usar el hint "parallel(table_name, default)" a nivel de objeto:
```
SELECT /*+ parallel(customers, default) */ COUNT(*) FROM customers;
```

Hay que tener en cuenta que la configuración de la tabla que se muestra arriba en el n.° 1 le proporciona el DOP predeterminado en modo manual, es decir, cuando PARALLEL_DEGREE_POLICY está configurado a MANUAL.

El paralelismo DEFAULT o por defecto, se dirige a una carga de trabajo monousuario y está diseñado para usar el máximo de recursos suponiendo que la operación finalizará más rápido si se usan más recursos. La base de datos no verifica si el paralelismo tiene sentido o si el paralelismo le proporcionará cualquier tipo de escalabilidad. Por ejemplo, puede ejecutar un SELECT * FROM emp; con el paralelismo predeterminado en el sistema descrito anteriormente, pero no verá ninguna escalabilidad al devolver estos 14 registros. En un entorno multiusuario, el paralelismo por defecto agotará rápidamente los recursos del sistema sin dejar recursos disponibles para que otros usuarios realicen ejecuciones en paralelo de forma simultánea.

Fixed Degree of Parallelism (DOP) ó grado de paralelismo fijo

A parte del paralelismo DEFAULT, se puede solicitar un DOP específico a Oracle Database.

Se puede usar una de las siguientes vías para utilizar un DOP fijo:

1. Indicar un DOP específico para los objetos:
```
ALTER TABLE customers PARALLEL 8;
ALTER TABLE sales PARALLEL 16;
```

2. Usar el hint "parallel(integer)" a nivel de sentencia:
```
SELECT /*+ parallel(8) */ COUNT(*) FROM customers;
```

3. Usar el hint "parallel(table_name, integer)" a nivel de objeto:
```
SELECT /*+ parallel(customers, 8) */ COUNT(*) FROM customers;
```

Hay que tener en cuenta que la configuración de la tabla que se muestra en el n.° 1 anterior solo le proporciona el DOP fijo en modo manual y modo limitado, es decir, cuando el PARALLEL_DEGREE_POLICY está establecido a MANUAL o LIMITED. En modo AutoDOP (AUTO o ADAPTIVE) se ignorará cualquier declaración de tabla.

Otra cosa a tener en cuenta del n.º 1 anterior es que las consultas que accedan a la tabla CUSTOMERS usarán un DOP de 8, las que accedan a la tabla SALES un DOP de 16 y las que accedan a ambas tablas usarán el mayor de los DOP; en este caso 16. Esta regla la cumplen todas las sentencias a excepción de las CTAS.

Grado de paralelismo automático (Auto DOP)

Con el Grado de Paralelismo Automático (Auto DOP), la base de datos decide automáticamente si una sentencia debe ejecutarse en paralelo o no y el DOP que debe usarse. La decisión de usar ejecución paralela y el DOP elegido se basa en los requisitos de recursos de la sentencia (aka "cost").

Si el tiempo transcurrido estimado para la sentencia es menor que PARALLEL_MIN_TIME_THRESHOLD, por defecto es AUTO que significa 10 segundos, la sentencia se ejecutará secuencialmente.

Si el tiempo transcurrido estimado es mayor que PARALLEL_MIN_TIME_THRESHOLD el optimizador utiliza el coste de todas las operaciones (full table scan, index fast full scan, aggregations, joins, etc) del plan de ejecución para determinar el DOP ideal para la sentencia. Oracle Database 12c usa los costes de CPU e IO de todas las operaciones mientras que la 11g usaba solo los costes de IO para el cálculo del Auto DOP.

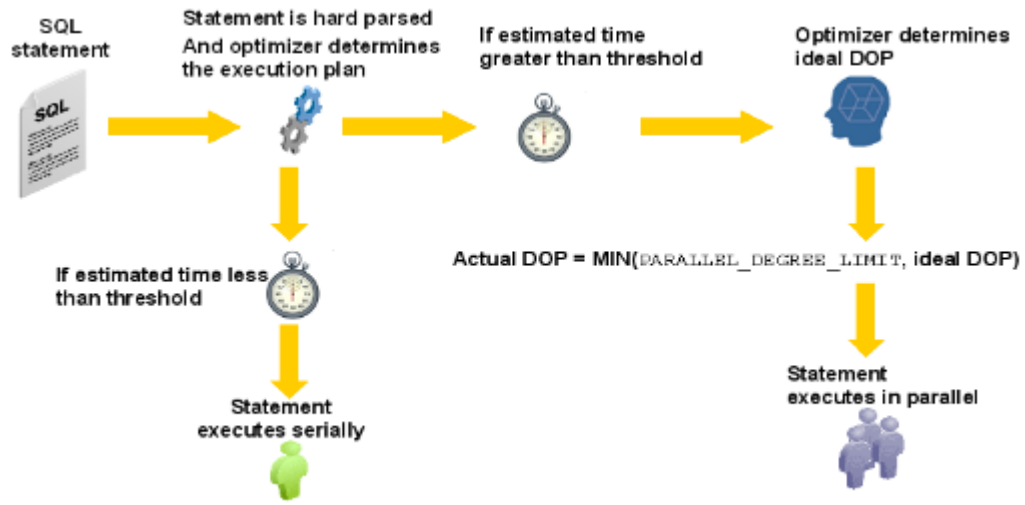
Dependiendo del coste de la sentencia, el DOP ideal puede llegar a ser muy grande. Para asegurarse de no asignar demasiados procesos de ejecución en paralelo a una única sentencia, el Optimizador limitará el DOP actual utilizado. Esta limitación se establece con el parámetro PARALLEL_DEGREE_LIMIT. El valor por defecto para este parámetro es CPU, que significa que el DOP está limitado por el número de CPUs del sistema. La fórmula usada para derivar el DOP predeterminado es:

```
PARALLEL_THREADS_PER_CPU * SUM(CPU_COUNT across all cluster nodes)
```

El optimizador comparará su DOP ideal con el PARALLEL_DEGREE_LIMIT y cogerá el valor más bajo.

```
ACTUAL DOP = MIN(IDEAL DOP, PARALLEL_DEGREE_LIMIT)
```

Es possible establecer PARALLEL_DEGREE_LIMIT a un número específico, permitiendole a usted controlar el DOP máximo que puede usar en su sistema.



El anterior diagrama muestra en detalle las decisiones de paralelización de una sentencia que se han tomado y el DOP que se ha usado.

El DOP final seleccionado se muestra y se explica en la sección de notes del plan de ejecución, que puede verse usando tanto el comando explain plan o V$SQL_PLAN. El plan que se muestra a continuación se generó en una única instancia de base de datos que corría en un servidor de 8 CPUs. En la sección de notes, puede ver que se ha sido seleccionado un DOP de 16. Este número, 16, es el DOP máximo permitido por PARALLEL_DEGREE_LIMIT en este sistema, 2 * 8.

```
SQL> EXPLAIN PLAN FOR
  2  SELECT SUM(l_quantity * l_extendedprice) total_rev
  3  FROM Lineitem;

Explained.

SQL> select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
Plan hash value: 1596562243

-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name     | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |          |     1 |     9 | 59931   (1)| 00:14:00 |       |       |        |      |            |
|   1 |  SORT AGGREGATE               |          |     1 |     9 |            |          |       |       |        |      |            |
|   2 |   PX COORDINATOR              |          |       |       |            |          |       |       |        |      |            |
|   3 |    PX SEND QC (RANDOM)        | :TQ10000 |     1 |     9 |            |          |       |       | Q1,00  | P->S | QC (RAND)  |
|   4 |     SORT AGGREGATE            |          |     1 |     9 |            |          |       |       | Q1,00  | PCWP |            |
|   5 |      PX BLOCK ITERATOR        |          |   479M|  4118M| 59931   (1)| 00:14:00 |     1 |    32 | Q1,00  | PCWC |            |
|   6 |       TABLE ACCESS STORAGE FULL| LINEITEM |   479M|  4118M| 59931   (1)| 00:14:00 |     1 |  2688 | Q1,00  | PCWP |            |
-------------------------------------------------------------------------------------------------------------------------------

Note
-----
   - automatic DOP: Computed Degree of Parallelism is 16 because of degree limit
```

El Auto DOP desde Oracle Database 11gR2 se controla por el parámetro de inicialización PARALLEL_DEGREE_POLICY, que en 12cR2 puede tener los valores {MANUAL | LIMITED | AUTO | ADAPTIVE}:

- MANUAL: El valor por defecto para PARALLEL_DEGREE_POLICY es MANUAL que significa que Auto DOP no está activo.

- LIMITED: Al establecer LIMITED, Auto DOP solo se aplica a sentencias que acceden a tablas o índices que añaden la cláusula PARALLEL pero sin un DOP

explícito. Las tablas o índices que tengan un DOP específico o fijo utilizarán ese DOP específico.

- AUTO: Es el valor recomendado para PARALLEL_DEGREE_POLICY. Al establecer AUTO, AUTO DOP se aplica a todas las sentencias SQL ejecutadas en el sistema, sin importar si acceden a objetos con la claúsula PARALLEL. Habilita el grado de paralelismo automático, la statement queuing y la ejecución paralela en memoria. Dado este comportamiento, es posible ejecutar en paralelo más sentencias SQL en Oracle Database 12c y 11g que en versiones anteriores.

- ADAPTIVE: Este valor habilita el grado de paralelismo automático, la statement queuing y la ejecución paralela en memoria; similar al valor AUTO. Además, el performance feedback está habilitado. El performance feedback ayuda a mejorar el grado de paralelismo elegido automáticamente para las sentencias SQL repetidas. Después de la ejecución inicial de una sentencia, el grado de paralelismo elegido por el optimizador se compara con el grado de paralelismo calculado en función del rendimiento de la ejecución real. Si varían significativamente, la sentencia se marca para volver parsear y las estadísticas de rendimiento de ejecución inicial (por ejemplo, CPU-time) se proporcionan como feedback para las ejecuciones posteriores. El optimizador usa las estadísticas de rendimiento de ejecución inicial para determinar mejor el grado de paralelismo para las ejecuciones posteriores.

Este parámetro de inicialización puede aplicarse a nivel de sistema o a nivel de sesión. Además, se puede invocar Auto DOP para una sentencia SQL específica usando el hint PARALLEL o PARALLEL(AUTO):

```
SELECT /*+ parallel(auto) */ COUNT(*) FROM customers;

SELECT /*+ PARALLEL (AUTO) */ SUM(l_quantity
*l_extendedprice) total_rev FROM Lineitem;
```

Paralelismo adaptativo (funcionalidad deprecada en 12.2)

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 garantizando una utilización óptima del sistema, los usuarios podrían experimentar tiempos de respuesta inconsistentes. 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, el valor predeterminado de este parámetro es false y 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.

Gestión de la carga de trabajo para ejecución paralela

Independientemente del patrón de carga de trabajo previsto querrá asegurarse de que las capacidades de ejecución paralela de Oracle se utilizan de la forma más óptima en su entorno. Esto implica 4 tareas fundamentales:

1. Controlar el grado de paralelismo
2. Asegurarse de que el sistema no se sobrecargue por procesamiento paralelo
3. Asegurarse de que cualquier sentencia determinada obtenga los recursos paralelos requeridos
4. Adherirse a las potenciales prioridades diferentes para diferentes grupos de usuarios

El framework AutoDOP de Oracle no solo controla el uso del procesamiento paralelo (1) de manera integral sin la intervención del usuario, sino que aborda los 3 primeros requisitos. Junto con Database Resource Manager, que es responsable del cuarto requisito, Oracle proporciona un framework integral de gestión de carga de trabajo para abordar los requisitos de carga de trabajo mixta más complejos del mundo.

Garantizar un DOP mínimo

Puede garantizarse un DOP mínimo usando el parámetro de inicialización PARALLEL_MIN_PERCENT. Este parámetro controla el porcentaje mínimo de procesos paralelos que deben estar disponibles para comenzar la operación; por defecto es 0, lo que significa que Oracle siempre ejecutará la sentencia, independientemente del nímero de procesos paralelos disponibles.

Por ejemplo, si desea garantizar que estén disponibles al menos un 50% de los procesos paralelos necesarios para una sentencia:

```
SQL> alter session set parallel_min_percent=50 ;
SQL> SELECT /*+ PARALLEL(s,128) */ count(*) FROM sales s;
SELECT /*+ PARALLEL(s,128) */ count(*) FROM sales s
                                            *
ERROR at line 1:
ORA-12827: insufficient parallel query slaves available
```

Si hubiera insuficientes procesos de consulta en paralelo disponibles – en este ejemplo menos de 64 procesos PX para una sentencia SQL (o menos de 128 esclavos para una operación compleja, que involucre productores y consumidores) – verá

ORA-12827 y la sentencia no se ejecutará. Puede capturar este error en su código y volver a intentarlo después.

Gestión del número de procesos de ejecuciones paralelas (PX)

Oracle Database asigna procesos a las operaciones paralelas desde un pool de procesos PX. El parámetro PARALLEL_MAX_SERVERS establece la cantidad máxima de procesos PX del pool. Este es un límite estricto que se usa para evitar sobrecargar el sistema con demasiados procesos. Por defecto, este parámetro está configurado de la siguiente forma:

```
5 * concurrent_parallel_users * CPU_COUNT * PARALLEL_THREADS_PER_CPU
```

El valor de `concurrent_parallel_users` se calcula de la siguiente forma:

- Si los parámetros de inicialización SGA_TARGET o MEMORY_TARGET están establecidos, entonces el número de `concurrent_parallel_users = 4`
- Si los parámetros de inicialización SGA_TARGET o MEMORY_TARGET no están establecidos, la PGA_AGGREGATE_TARGET es examinada. Si este parámetro tiene un valor definido, entonces `concurrent_parallel_users = 2`; si no, `concurrent_parallel_users = 1`

Cuando todos los procesos del pool han sido asignados, las nuevas operaciones que requieren paralelismo se ejecutan de forma serializada o con un DOP degradado que causa un rendimiento degradado para estas operaciones.

Para evitar llegar al PARALLEL_MAX_SERVERS y serializar o degradar las operaciones, Auto DOP utiliza un límite adicional establecido por el parámetro PARALLEL_SERVERS_TARGET. Por defecto, este parámetro está configurado a:

```
2 * concurrent_parallel_users * CPU_COUNT * PARALLEL_THREADS_PER_CPU
```

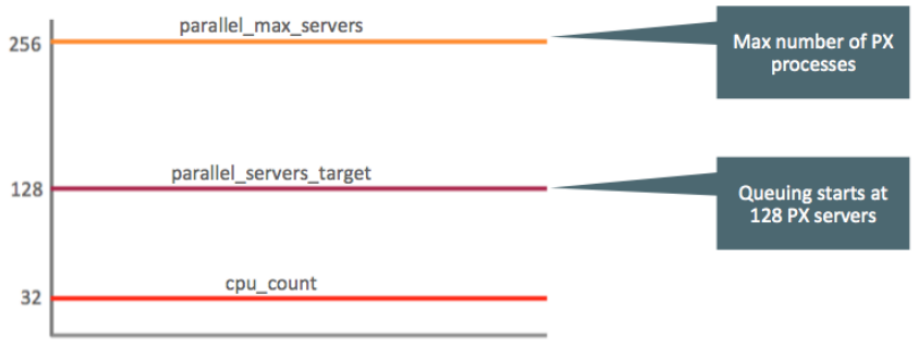


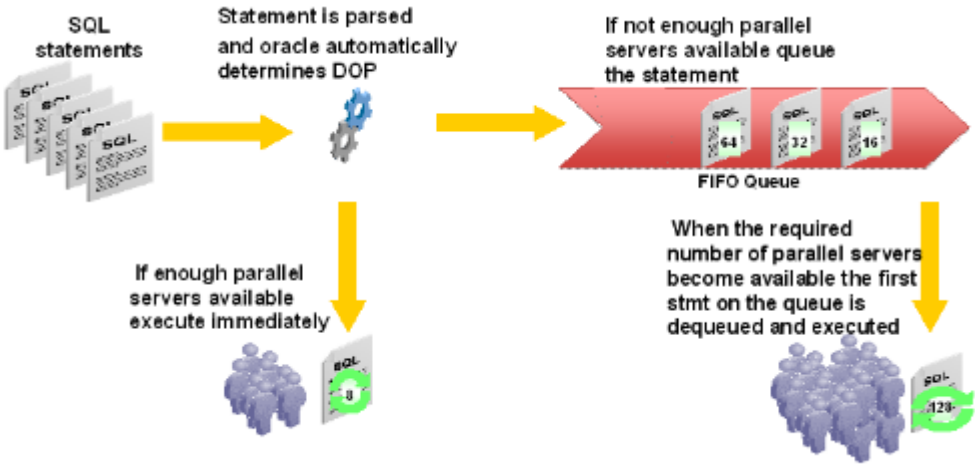
Figure : A sample configuration showing the limits for the number of PX server processes

PARALLEL_SERVERS_TARGET es la cantidad de procesos PX disponibles para ejecutar sentencias paralelas antes de que se use el encolamiento de sentencias. Se establece en un valor inferior al de PARALLEL_MAX_SERVERS para garantizar que cada sentencia paralela obtenga todos los recursos necesarios del proceso PX y para evitar sobrecargar el sistema con procesos PX. Hay que tener en cuenta que todas las sentencias serializadas (no paralelas) se ejecutarán inmediatamente aunque se haya activado la cola de sentencias. El límite inferior de PARALLEL_SERVERS_TARGET solo se tiene en cuenta cuando se ejecuta con PARALLEL_DEGREE_POLICY en

AUTO o ADAPTIVE, la configuración de inicialización obligatoria para invocar la funcionalidad completa de Auto DOP.

Encolamiento de sentencias

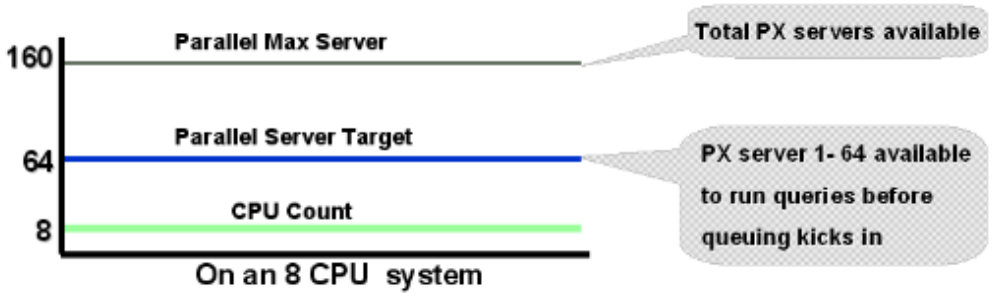
A través de Statement Queuing, Oracle encolará sentencias SQL que necesiten ejecución paralela si el número de procesos PX especificado no están disponibles. Una vez que los recursos necesarios estén disponibles, la sentencia SQL irá desencolando y permitiendo la ejecución. Con este mecanismo, Oracle garantiza que las sentencias se ejecutan con el DOP especificado.



La cola de sentencias es una cola primero en entrar – primero en salir (FIFO) basada en el tiempo en el que fue lanzada la sentencia SQL. El mecanismo de Statement Queueing se iniciará en cuanto el número de procesos paralelos activos en el sistema sea mayor o igual que PARALLEL_SERVERS_TARGET. Por defecto, este parámetro se establece a

```
4 * CPU_COUNT * PARALLEL_THREADS_PER_CPU * ACTIVE_INSTANCE_COUNT
```

PARALLEL_SERVERS_TARGET no es el número máximo de procesos PX permitidos en el sistema sino el número de procesos disponibles para ejecutar sentencias en paralelo antes de usar Statement Queueing. Se establece a un valor más bajo que el número de procesos PX permitidos en el sistema, controlado por PARALLEL_MAX_SERVERS, para garantizar que cada sentencia paralela obtenga todos los procesos PX requeridos y así prevenir la sobrecarga del sistema con procesos PX. Considere que todas las sentencias secuenciales (no paralelas) se ejecutarán inmediatamente incluso si statement queue está activado.



Puede identificar qué sentencias SQL se están encolando usando [GV|V]$SQL_MONITOR o la pantalla SQL MONITOR en Oracle Enterprise Manager (EM).

```
SELECT sql_id, sql_text
FROM [GV|V]$SQL_MONITOR
WHERE status='QUEUED';
```

Hay también dos eventos de espera para saber si una sentencia ha sido encolada. Una sentencia esperando en el evento "PX QUEUING: statement queue" es la primera sentencia en la cola de sentencias. Una vez que los recursos necesarios esten disponibles para esta sentencia, se desencolará y se ejecutará. Todas las demás sentencias en la cola estarán esperando en "ENQ JX SQL statement queue". Solo cuando una sentencia sea la primera en la cola el evento de espera cambiará a "PXQUEUING: statement queue".

Es possible evitar o saltarse la restricción del mecanismo de Statement Queueing usando NO_STMT_QUEUING.

```
SELECT /*+ NO_STMT_QUEUING */
SUM(l_quantity *l_extendedprice) total_rev
FROM Lineitem;
```

Statement Queueing se activa solo cuando el parámetro PARALLEL_DEGREE_POLICY se establece a AUTO. Si quiere asegurar que una sentencia no se ejecute hasta que pueda obtener todos los procesos paralelos que necesita cuando statement queue no está activa puede usar STMT_QUEUING.

Uso del Paquete DBMS_PARALLEL_EXECUTE

El paquete DBMS_PARALLEL_EXECUTE provee subprogramas para permitir que un determinado INSERT, UPDATE, DELETE, MERGE, o un bloque de sentencias, pueda ser aplicado en trozos paralelos. La sentencia debe tener dos marcadores de posición que definen el límite de inicio y final de un trozo. Por lo general, estos son valores como el ROWID o un número de clave única (PK), de una gran tabla. No obstante, cuando un bloque se sentencias se utiliza, el bloque puede interpretar los valores arbitrariamente. El paquete cuenta con subprogramas para definir los rangos que cubren la tabla especificada. Estos incluyen, un método basado en reglas para dividir la tabla por ROWID o número de clave única, o un método definido por el usuario. La sentencia SQL junto con el conjunto de rangos de trozos definien una tarea. Otro subprograma inicia la tarea. Cada tarea se procesa utilizando un programador de trabajos y automáticamente se valida (commit) cuando se completa. Otro subprograma, permite que la tarea sea reanudada, si ocurrió un error en su ejecución.

Etapas del uso del paquete:

1. **Creación de una tarea:**

```
BEGIN
  DBMS_PARALLEL_EXECUTE.create_task (task_name =>
'test_task');
END;
```

```
/
```

Para ver las tareas creadas:

```
SELECT task_name,
       status
FROM   user_parallel_execute_tasks;
```

2. **División de la carga en trozos (rangos):**

CREATE_CHUNKS_BY_ROWID:
Si el parámetro `BY_ROW` = TRUE, el `CHUNK_SIZE` se refiere a números de registros, si el FALSE, se refiere a número de bloques:

```
BEGIN
  DBMS_PARALLEL_EXECUTE.create_chunks_by_rowid(
  task_name   => 'test_task',
  table_owner => 'TEST',
  table_name  => 'TEST_TAB',
  by_row      => TRUE,
  chunk_size  => 10000);
END;
/
```

CREATE_CHUNKS_BY_NUMBER_COL:
Este procedimiento divide la carga de trabajo según el número de una columna, con números mínimos y máximos.

```
BEGIN
  DBMS_PARALLEL_EXECUTE.create_chunks_by_number_col(
  task_name    => 'test_task',
  table_owner  => 'TEST',
  table_name   => 'TEST_TAB',
  table_column => 'ID',
  chunk_size   => 10000);
END;
/
```

CREATE_CHUNKS_BY_SQL:
Este procedimiento divide la carga de trabajo según una consulta definida por el usuario. Si el parámetro `BY_ROWID` = TRUE la SQL retorna una serie de principio y final de rowids, si es FALSE la SQL retorna una serie de principio y final de IDs:

```
DECLARE
  l_stmt CLOB;
BEGIN
  l_stmt := 'SELECT DISTINCT num_col, num_col FROM
  test_tab';

  DBMS_PARALLEL_EXECUTE.create_chunks_by_sql(
  task_name => 'test_task',
  sql_stmt  => l_stmt,
  by_rowid  => FALSE);
END;
/
```

Para ver información de cada trozo:

```
SELECT chunk_id, status, start_rowid, end_rowid
FROM   user_parallel_execute_chunks
WHERE  task_name = 'test_task'
ORDER BY chunk_id;
```

3. **Ejecución de la tarea:**

```
DECLARE
  l_sql_stmt VARCHAR2(32767);
BEGIN
  l_sql_stmt := 'UPDATE test_tab t
                 SET    t.num_col = t.num_col + 10,
                        t.session_id =
SYS_CONTEXT(''USERENV'',''SESSIONID'')
                 WHERE rowid BETWEEN :start_id AND
:end_id';

  DBMS_PARALLEL_EXECUTE.run_task(
  task_name => 'test_task',
  sql_stmt => l_sql_stmt,
  language_flag  => DBMS_SQL.NATIVE,
  parallel_level => 10);
END;
/
```

Para ver la actividad de la ejecución en paralelo:

```
SELECT session_id, COUNT(*)
FROM   test_tab
GROUP BY session_id
ORDER BY session_id;
```

4. **Comprobación del estado de la tarea:**

```
COLUMN task_name FORMAT A10
SELECT task_name,
       status
FROM   user_parallel_execute_tasks;

SELECT status, COUNT(*)
FROM   user_parallel_execute_chunks
GROUP BY status
ORDER BY status;

SELECT job_prefix
FROM   user_parallel_execute_tasks
WHERE  task_name = 'test_task';

COLUMN job_name FORMAT A20

SELECT job_name, status
FROM   user_scheduler_job_run_details
WHERE  job_name LIKE (SELECT job_prefix || '%'
                      FROM   user_parallel_execute_tasks
                      WHERE  task_name = 'test_task');
```

Si después de la ejecución el estado es distinto de FINISHED, entonces la tareas puede ser reanudada con el procedimiento RESUME_TASK:

```
DECLARE
  l_try NUMBER;
  l_status NUMBER;
BEGIN
  -- If there is error, RESUME it for at most 2 times.
  l_try := 0;
  l_status :=
DBMS_PARALLEL_EXECUTE.task_status('test_task');
  WHILE(l_try < 2 and l_status !=
DBMS_PARALLEL_EXECUTE.FINISHED)
  Loop
    l_try := l_try + 1;
    DBMS_PARALLEL_EXECUTE.resume_task('test_task');
    l_status :=
DBMS_PARALLEL_EXECUTE.task_status('test_task');
  END LOOP;
END;
/
```

5. **Eliminación de la tarea:**

```
BEGIN
  DBMS_PARALLEL_EXECUTE.drop_task('test_task');
END;
/
```

Muchos escenarios requieren una gran transformación de un gran número de filas. Utilizando una sentencia SQL ordinaria, podemos sufrir el efecto todo o nada. En el caso común, donde la transformación de una fila es independiente de la de otras filas, es correcto validar (commit) cada fila que se transforma con éxito y revertir todas las filas donde la transformación falla.

Gestión de recursos con Oracle Database Resource Manager (DBRM)

Oracle Database Resource Manager (DBRM) permite priorizar el trabajo en Oracle Database y restringir el acceso a recursos para determinados grupos de usuarios. Es altamente recomendable usar DBRM si un sistema tiene límite de CPU, ya que protege a los usuarios o tareas con prioridad alta de ser impactados por tareas con prioridad menor. Proporciona esta protección asignando tiempo de CPU a las diferentes tareas en base a su prioridad. Para usar DBRM necesita crear grupos de consumidores, que son grupos de usuarios con determinadas características, por ejemplo username o rol. Puede crear un plan de recursos donde se especifique cómo tienen que distribuirse los recursos entre los diversos grupos de consumidores.

Los recursos incluyen porcentaje de tiempo de CPU, número de sesiones activas, y cantidad de espacio disponible en el tablespace undo. También puede restringir la ejecución paralela para los usuarios de un grupo de consumidores. DBRM es el factor decisivo final para determinar el grado máximo de paralelismo, y ningún usuario en un grupo de consumidores (usando un plan de recursos específico) será capaz de ejecutar con un DOP mayor que el máximo del grupo de recursos.

Por ejemplo, si su plan de recursos tiene una política de usar un DOP máximo de 4 y solicita un DOP de 16, su SQL se ejecutará con un DOP de 4. Esto es cierto independientemente de cómo fue obtenido el DOP de 16, tanto si se especificó como un atributo de la tabla o si se obtuvo mediante las capacidades de AUTO DOP de Oracle.

Además, DBRM puede controlar el número máximo de sesiones activas para un determinado grupo de recursos. Así para el plan de recursos mostrado DW_USERS se permiten un máximo de 4 sesiones activas, resultando un total de consumo máximo de recursos de 4 (sesiones) x 4 (DOP) x 2 (conjuntos de esclavos) = 32 procesos PX.

Inicialización de parámetros que controlan la ejecución paralela

Se pueden usar varios parámetros de inicialización para controlar el comportamiento de las ejecuciones en paralelo en la base de datos. Sin embargo, los fundamentales son los siguientes:

- PARALLEL_DEGREE_POLICY: {MANUAL | LIMITED | AUTO | ADAPTIVE} controla si están o no habilitados In-Memory PX (ejecución paralela en memoria), Auto DOP o Statement Queuing. El valor por defecto es MANUAL en 11g y 12c, que deshabilita funcionalidad y el valor recomendado es AUTO:
  - MANUAL: El valor por defecto para PARALLEL_DEGREE_POLICY es MANUAL que significa que Auto DOP no está activo.
  - LIMITED: Al establecer LIMITED, Auto DOP solo se aplica a sentencias que acceden a tablas o índices que añaden la cláusula PARALLEL pero sin un DOP explícito. Las tablas o índices que tengan un DOP específico o fijo utilizarán ese DOP específico.
  - AUTO: Es el valor recomendado para PARALLEL_DEGREE_POLICY. Al establecer AUTO, AUTO DOP se aplica a todas las sentencias SQL ejecutadas en el sistema, sin importar si acceden a objetos con la claúsula PARALLEL. Habilita el grado de paralelismo automático, la statement queuing y la ejecución paralela en memoria. Dado este comportamiento, es posible ejecutar en paralelo más sentencias SQL en Oracle Database 12c y 11g que en versiones anteriores.
  - ADAPTIVE: Este valor habilita el grado de paralelismo automático, la statement queuing y la ejecución paralela en memoria; similar al valor AUTO. Además, el performance feedback está habilitado.

  Más detalle en el sub-apartado *Grado de paralelismo automático (Auto DOP)* de este mismo informe y en PARALLEL_DEGREE_POLICY.

- PARALLEL_SERVERS_TARGET: Este parámetro especifica el número de procesos paralelos permitidos para ejecutarse antes de que se use el statement queuing y solo está activo cuando el parámetro PARALLEL_DEGREE_POLICY está establecido en AUTO o ADAPTIVE, Oracle pondrá en cola las sentencias SQL que requieran ejecución paralela si el número requerido de procesos PX no están disponibles. El encolamiento de sentencias comenzará una vez que la cantidad de procesos PX activos en el sistema sea igual o mayor que el PARALLEL_SERVERS_TARGET. El valor predeterminado para este parámetro

es el 40% de PARALLEL_MAX_SERVERS, lo que garantiza que haya un búfer de recursos para otras operaciones, como las sentencias paralelas que pasan por alto el statement queuing. Se recomienda establecer este parámetro en un valor que sea necesario para sus operaciones paralelas gestionadas por Auto DOP teniendo en cuenta los recursos del sistema asignados disponibles para el procesamiento paralelo. Más detalle en PARALLEL_SERVERS_TARGET.

- PARALLEL_DEGREE_LIMIT: Con el grado automático de paralelismo, Oracle decide automáticamente si una sentencia debe ejecutarse en paralelo y qué grado de paralelismo debe usar. El optimizador determina automáticamente el grado de paralelismo para una sentencia en función de los requisitos de recursos de la propia sentencia. Sin embargo, el optimizador limitará el grado de paralelismo utilizado para garantizar que los procesos de ejecuciones paralelas no sobrecarguen el sistema. Este límite lo define el parámetro PARALLEL_DEGREE_LIMIT. Por defecto, este parámetro está configurado para CPU, lo que limita el DOP máximo al DOP por defecto del sistema. Se recomienda dejar este parámetro a su valor por defecto y usar Database Resource Manager para un control detallado del DOP máximo. Más detalle en el sub-apartado *Grado de paralelismo automático (Auto DOP)* de este mismo informe y en PARALLEL_DEGREE_LIMIT.

- PARALLEL_MAX_SERVERS: el número máximo de procesos PX que pueden ponerse en marcha en una instancia de base de datos. Para ejecutar una operación en paralelo, los procesos PX deben estar disponibles (por ejemplo no estar en uso por otra operación paralela). Por defecto el valor de PARALLEL_MAX_SERVERS se obtiene por las características de la base de datos y normalmente es (`PARALLEL_THREADS_PER_CPU * CPU_COUNT * concurrent_parallel_users * 5`). Volviendo atrás al ejemplo de contar coches usando la ayuda de amigos: PARALLEL_MAX_SERVERS es el número máximo de amigos que puede llamar para que le ayuden. Más detalle en el sub-apartado *Gestión del número de procesos de ejecuciones paralelas (PX)* de este mismo informe y en PARALLEL_MAX_SERVERS.

- PARALLEL_MIN_SERVERS: el número mínimo de procesos PX que siempre se ponen en marcha cuando la instancia de la base de datos se está ejecutando. PARALLEL_MIN_SERVERS permite evitar o retrasar la ejecución de una operación paralela intercambiando los procesos PX. Por defecto el valor se establece con (`CPU_COUNT * PARALLEL_THREADS_PER_CPU * 2`). En el ejemplo de contar coches PARALLEL_MIN_SERVERS es el número de amigos que están allí con usted y que no tiene que llamar para comenzar el trabajo de contar coches. Más detalle en PARALLEL_MIN_SERVERS.

La tabla mostrada a continuación da una breve explicación de los otros parámetros de inicialización para ejecuciones paralelas:

Nombre del parámetro	Valor por defecto	Explanación
PARALLEL_FORCE_LOCAL	FALSE	En un entoro RAC controla si un proceso paralelo se lanza en un único nodo o no
PARALLEL_INSTANCE_GROUP	N/A	Si se usa junto al parámetro INSTANCE_GROUPS, restringe las operaciones de consulta paralela a un número limitado de instancias
PARALLEL_MIN_PERCENT	0	Mínimo porcentaje de procesos paralelos necesarios para la ejecución paralela.
PARALLEL_MIN_TIME_THRESHOLD	AUTO	Tiempo de ejecución mínima que puede tener una sentencia antes de que AUTO DOP se active
PARALLEL_THREADS_PER_CPU	2	Número de procesos paralelos que una CPU puede manejar durante la ejecución paralela
PARALLEL_EXECUTION_MESSAGE_SIZE	16KB	Tamaño de los buffers utilizados por los procesos de ejecución paralela para comunicarse con los demás y el QC

Parámetros deprecados en Oracle Database 12cR2

En Oracle Database 12.2 se ha deprecado la funcionalidad Adaptive Parallelism que se controla con el parámetro PARALLEL_ADAPTIVE_MULTI_USER.

Nombre del parámetro	Valor por defecto	Explanación
PARALLEL_ADAPTIVE_MULTI_USER	FALSE	Acelera el DOP de una sentencia SQL basándose en la carga de trabajo concurrente. Puede provocar tiempos de respuesta no deterministas.

Parámetros obsoletos en Oracle Database 12cR2

Parámetros obsoletos y por lo tanto, eliminados en Oracle Database 12cR2. Estos parámetros fueron deprecados hace tiempo pero han estado presentes hasta la 12cR1; en 12cR2 han sido eliminados:

- PARALLEL_SERVER
- PARALLEL_SERVER_INSTANCES
- PARALLEL_IO_CAP_ENABLED
- PARALLEL_AUTOMATIC_TUNING