Oracle 12c Parallel Execution New Features

Randolf Geist
Unabhängiger Berater
Heidelberg, Deutschland

Schlüsselworte
Oracle 12c, New Features, Parallel Execution

Einleitung

Oracle 12c bringt viele Neuerungen mit sich, auch in Bereichen, in denen es seit einigen Releases keine signifikanten Veränderungen gab. Das gilt vor allem für einige Bereiche des sogenannten „Parallel Execution" Features – eine Option der Enterprise Edition, die es erlaubt, eine SQL Ausführung automatisch parallelisiert von mehreren Prozessen gleichzeitig verarbeiten zu lassen.

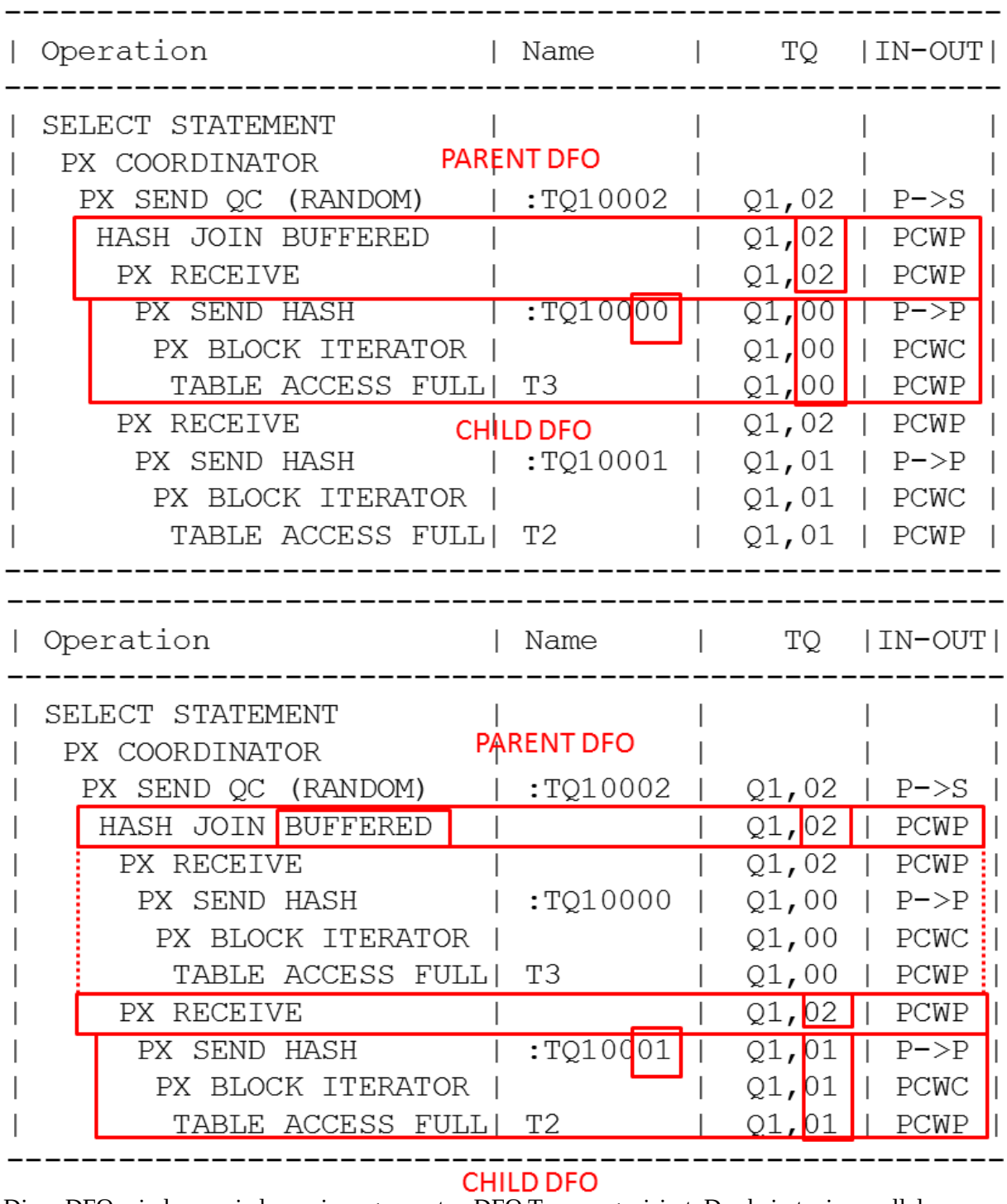
Hintergrund – wichtige Eigenschaften der parallelen Ausführung von SQL-Statements

Um die mit Oracle 12c eingeführten Neuerungen entsprechend einordnen zu können, soll hier auf drei wesentliche Eigenschaften der Parallel-Ausführung von SQL-Statements eingegangen werden.

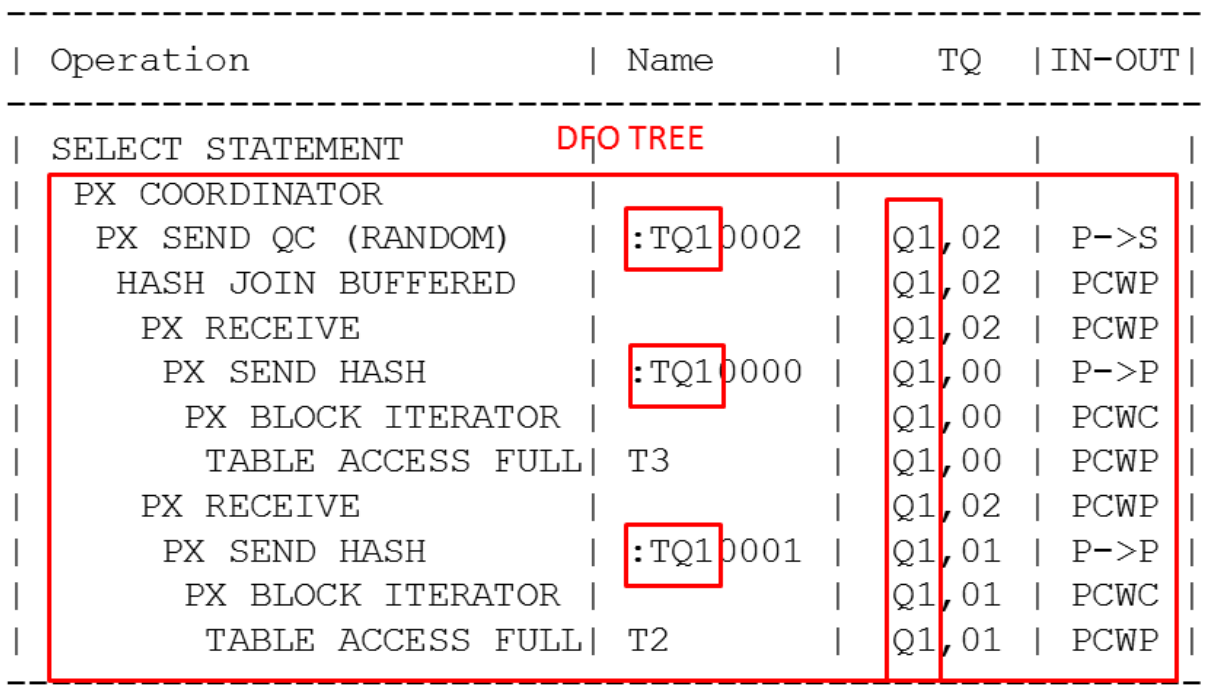
Producer-Consumer Modell

Die erste wesentliche Eigenschaft bei der Parallel-Ausführung von SQL-Statements ist der Einsatz des sogenannten Producer-Consumer Modells. Das heisst, um entsprechende relationale Operationen wie zum Beispiel Joins parallel ausführen zu können, verwendet Oracle in den meisten Fällen zwei Sets von Parallel Execution Servern bei der Ausführung, die Daten untereinander austauschen. Bei einem angenommenen Parallelitätsgrad von 8 als Beispiel, also im Grunde die Verteilung der SQL-Ausführung auf 8 Prozesse gleichzeitig, verwendet Oracle dann zweimal 8 Prozesse für die Ausführung - insgesamt also 16 Prozesse - die in zwei Sets mit je 8 Prozessen aufgeteilt sind. Diese zwei Sets kommunizieren dann über entsprechende logische und physische Kanäle miteinander.

Für diese Kommunikation im Rahmen des Producer-Consumer Modells teilt Oracle die Operationen eines parallelen Ausführungsplans in sogenannte „Data Flow Operations (DFOs)" auf. Diese DFOs bilden im Ausführungsplan eine Baum-Struktur mit Eltern- und Kind DFOs. Jeweils ein DFO-Paar wird dann von den zwei Sets an Parallel Execution Servern ausgeführt, und zwischen diesen zwei DFOs findet dann der entsprechende Datenaustausch statt.

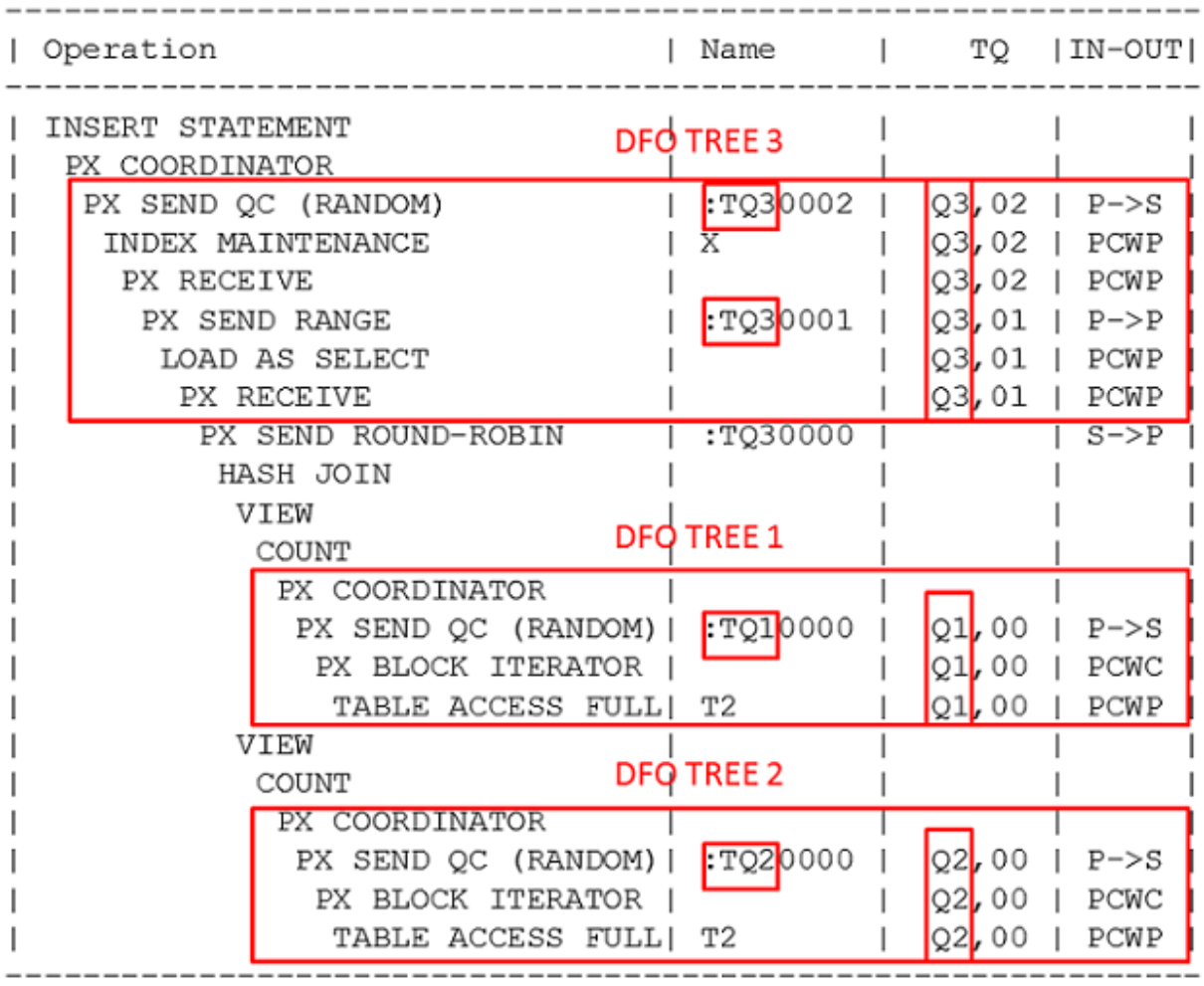


Diese DFOs sind nun wiederum in sogenannten DFO Trees organisiert. Das heisst, ein paralleler Ausführungsplan besteht aus mindestens einem DFO Tree, kann aber auch aus mehreren DFO Trees bestehen. Idealerweise sollte es in einem parallelen Ausführungsplan nur einen DFO Tree geben, aber je nach Einsatz und Kombination von bestimmten SQL Features muss Oracle aufgrund von internen Limitierungen den Ausführungsplan in mehrere DFO Trees aufteilen. Diese Aufteilung auf mehrere DFO Trees hat bestimmte Konsequenzen.



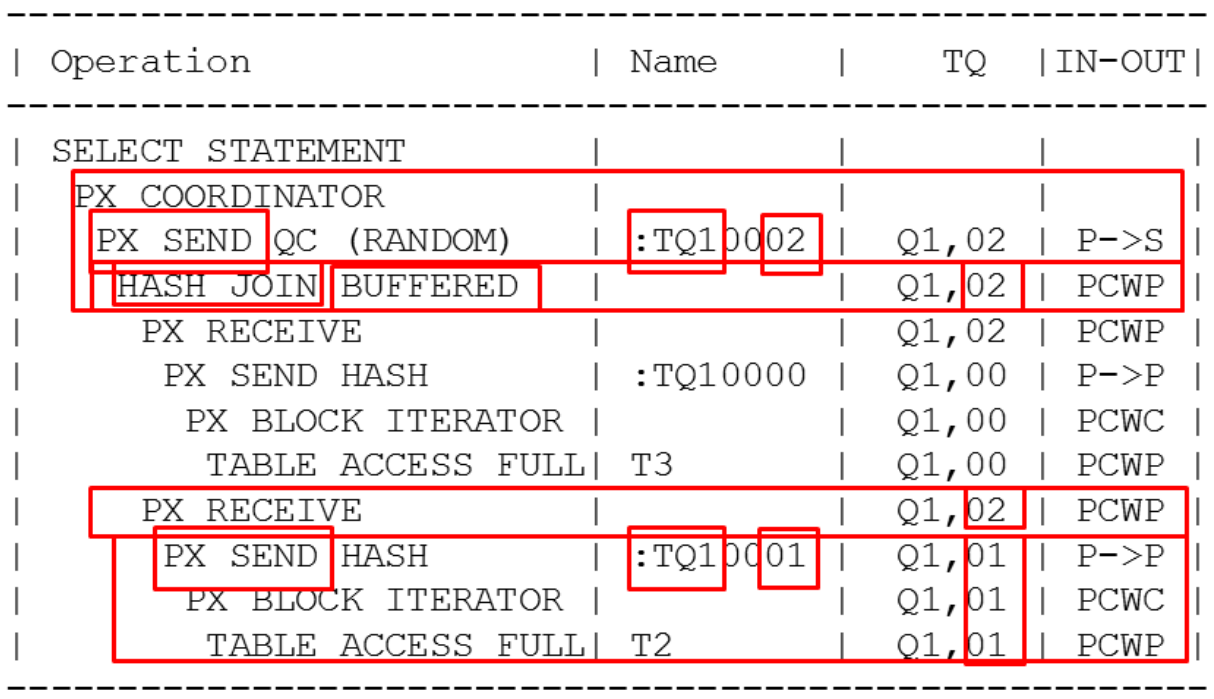
Zum Beispiel besteht jeder DFO Tree aus einer seriellen Start-Operation (PX COORDINATOR). Besteht ein Ausführungsplan also aus mehreren DFO Trees, müssen die Daten anstatt am Ende möglicherweise während der Verarbeitung durch solche seriellen Verarbeitungsteile des Ausführungsplans. Je nachdem, welche Operationen und welche Datenmengen hier verarbeitet werden, kann dies die zugrundeliegende Idee der parallelen Ausführung stark beeinträchtigen, so dass sehr viel Verarbeitungszeit in diesen seriellen Teilen des Ausführungsplans verbracht werden kann.

Darüber hinaus werden die Parallel Execution Server pro DFO Tree allokiert und de-allokiert, was mehrere Konsequenzen hat. Zum einen kann dies dazu führen, dass (wesentlich) mehr Parallel Execution Server gleichzeitig aktiv sind, als man aufgrund des Producer-Consumer Modells erwarten würde, also mehr als zwei Sets, zum anderen können tatsächlich pro DFO Tree unterschiedliche Parallelitätsgrade zum Einsatz kommen, was bedeutet, dass ein paralleler Ausführungsplan nicht unbedingt nur einen Parallelitätsgrad verwendet, sondern mehrere, nämlich pro DFO Tree. Außerdem kann je nach Ausführungsplan ein DFO Tree mehrfach gestartet werden, und bei jedem Start/Ende werden die Parallel Execution Server für diesen DFO Tree allokiert und wieder freigegeben. Bei häufigen Starts kann Oracle sehr viel Zeit nur mit dieser Allokierung und Freigabe verbringen, anstatt mit der eigentlichen Ausführung des SQLs.



Zusätzliche BUFFER-Operationen bei Parallelverarbeitung

Die zweite wesentliche Eigenschaft der Parallel-Ausführung ist, dass bei Oracle der Datenaustausch zwischen DFOs (also die Implementierung des Producer-Consumer Modells) nur für ein Paar DFOs pro DFO Tree gleichzeitig möglich ist, also nur genau ein Eltern- und Kind DFO gleichzeitig miteinander kommunizieren können. Wenn aber die Form des Ausführungsplans dazu führen würde, dass mehrere solche Kommunikationen gleichzeitig notwendig wären, was passiert dann? Oracle setzt dann tatsächlich zusätzliche, künstliche Operatoren ein, die die Daten zwischenspeichern müssen, damit eben immer nur ein Datenaustausch zwischen DFOs gleichzeitig stattfindet. Diese Notwendigkeit des Zwischenpufferns der Daten kann zu massivem zusätzlichem Bedarf an PGA-Speicher führen, der in vielen Fällen nicht ausreicht und daher auf den TEMP-Tablespace ausgelagert wird. Das heisst, durch dieses künstliche, temporäre Vorhalten der Daten kann der PGA bzw. TEMP-Bedarf von parallelen SQL-Ausführungen deutlich höher sein als von entsprechenden seriellen Ausführungsplänen.

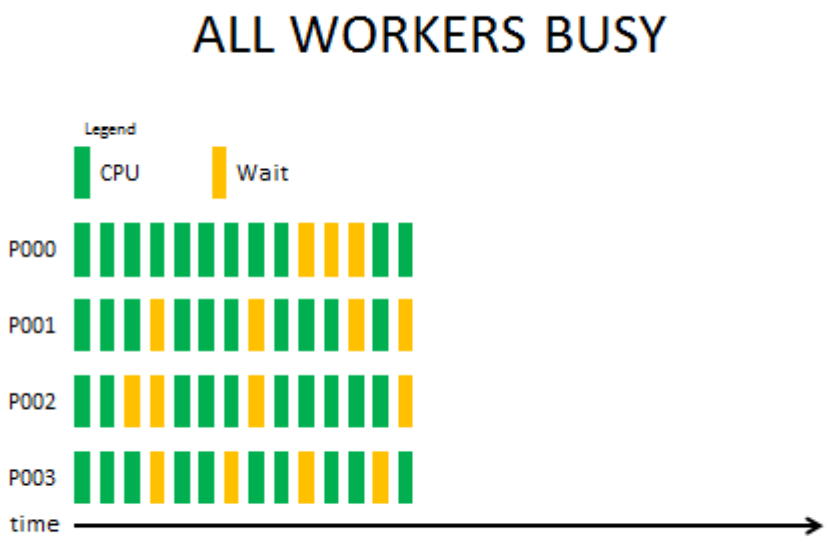


Operation	Name	TQ	IN-OUT
SELECT STATEMENT			
PX COORDINATOR			
PX SEND QC (RANDOM)	:TQ10002	Q1,02	P->S
HASH JOIN BUFFERED		Q1,02	PCWP
PX RECEIVE		Q1,02	PCWP
PX SEND HASH	:TQ10000	Q1,00	P->P
PX BLOCK ITERATOR		Q1,00	PCWC
TABLE ACCESS FULL	T3	Q1,00	PCWP
PX RECEIVE		Q1,02	PCWP
PX SEND HASH	:TQ10001	Q1,01	P->P
PX BLOCK ITERATOR		Q1,01	PCWC
TABLE ACCESS FULL	T2	Q1,01	PCWP

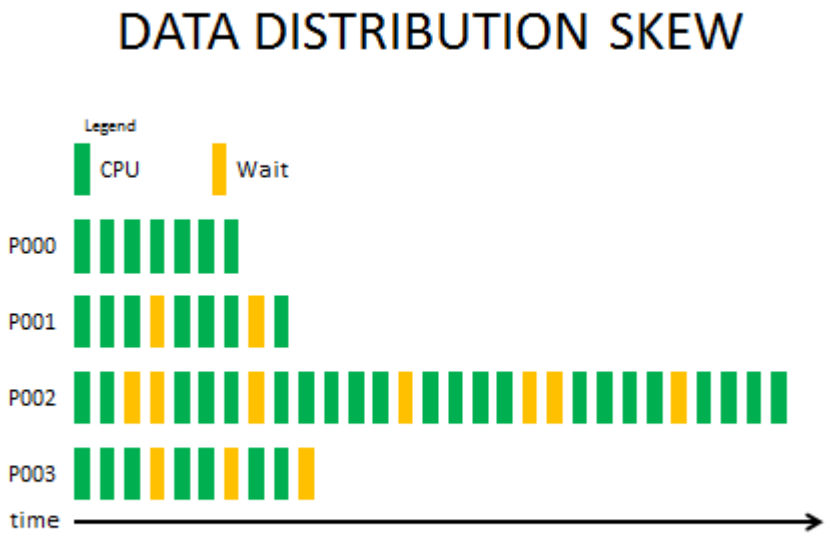
Potentielle Datenungleichverteilung bei Parallelverarbeitung

Die dritte wesentliche Eigenschaft der parallelen SQL-Ausführungen hängt damit zusammen, wie die Daten automatisch zwischen den Parallel Execution Servern aufgeteilt werden, um eine effiziente Parallelverarbeitung zu ermöglichen. Oracle kennt hier verschiedene Aufteilungsmethoden, und je nach verwendeter Methode und Daten kann es passieren, dass die Daten nicht gleichmäßig verteilt werden. Dies ist ein recht häufiges Phänomen bei paralleler Verarbeitung und schränkt die tatsächliche Parallelisierung unter Umständen massiv ein, da eventuell ein Großteil der Daten von wenigen Parallel Execution Servern verarbeitet werden müssen, während die anderen ihre (kleinere) Datenmenge schon lange verarbeitet haben und daher auf die anderen, noch beschäftigten Parallel Execution Servern warten müssen.

Eine graphische Darstellung einer optimalen Arbeitsverteilung könnte wie folgt aussehen. Beispielhaft hier für vier Parallel Execution Server gezeigt fangen alle zur gleichen Zeit an zu arbeiten, verbringen ungefähr die gleiche Zeit mit der Verarbeitung von Daten und sind zur gleichen Zeit fertig.



Die gleiche theoretische Arbeit könnte aber auch so verteilt werden:



Hier muss ein Parallel Execution Server wesentlich mehr Daten verarbeiten, während die anderen drei entsprechend weniger zu tun haben. Das Resultat dieser Ungleichverteilung ist, dass für einen längeren Zeitraum nur ein Prozess arbeitet, während die anderen schon fertig sind. Die

Gesamtausführungszeit wird aber durch den langsamsten Prozess bestimmt und entsprechend verlängert sich die Ausführungszeit hier, auch wenn insgesamt nicht mehr gearbeitet wurde.

Wir werden die mit Oracle 12c neuen Parallel Execution Features auch in Hinblick auf diese genannten drei Eigenschaften untersuchen.

Oracle 12c neue Parallel Execution Features

Neue Verteilungsmethode PX SEND HYBRID HASH

Wenn keine geeignete Partitionierung eingesetzt wird, waren vor Oracle 12c die zwei Standard-Verteilungsmethoden zur Aufteilung der Daten zwischen den Parallel Execution Servern BROADCAST und HASH. Bei BROADCAST werden die Daten tatsächlich pro Parallel Execution Server dupliziert, während bei HASH die Daten gemäß einer Hash-Funktion verteilt werden. BROADCAST ist also im Sinne der Parallel-Verarbeitung und der dafür eigentlich notwendigen Aufteilung der Daten unsinnig aufgrund der Datenduplizierung, kann aber dann Sinn machen, wenn die zu verteilende Datenmenge in Relation zur zweiten zu verteilenden Datenmenge (bei Joins sind immer zwei Datenquellen zu betrachten) recht klein ist, und man sich dann dafür die Verteilung der größeren, zweiten Datenquelle sparen kann, da die Verteilung zum einen auch Ressourcen benötigt (CPU und eventuell Netzwerk bei RAC), zum anderen auch immer das Potential zur Ungleichverteilung der Daten besteht. Außerdem wird durch die Reduzierung der Anzahl der Umverteilungen die Notwendigkeit der oben beschriebenen Zwischenpufferung potentiell verringert.

Die HASH-Verteilung kommt immer dann zum Einsatz, wenn die zwei Datenquellen recht ähnliche Größen haben, so dass ein BROADCAST einer Datenquelle keinen Sinn macht. Die Hash-Funktion liefert bei einigermaßen gleichmäßig verteilter Eingabewerte (Für Joins ist das die Join Expression) eine gute Verteilung der Daten auf die Parallel Execution Server, kann aber bei entsprechender Ungleichverteilung der Eingabewerte (also bei Joins zum Beispiel populäre Fremdschlüssel-Werte, die wesentlich häufiger vorkommen als andere) auch zu einer starken Ungleichverteilung der Daten führen, so dass eben oben beschriebener Effekt der ungleichen Verteilung der Arbeit auf die Parallel Execution Server auftreten kann.

```
-----------------------------------------------------------
| Operation                | Name      |    TQ   |IN-OUT|
-----------------------------------------------------------
| SELECT STATEMENT         |           |         |      |
|  PX COORDINATOR          |           |         |      |
|   PX SEND QC (RANDOM)    | :TQ10002  |  Q1,02  | P->S |
|    HASH JOIN BUFFERED    |           |  Q1,02  | PCWP |
|     PX RECEIVE           |           |  Q1,02  | PCWP |
|      PX SEND HASH        | :TQ10000  |  Q1,00  | P->P |
|       PX BLOCK ITERATOR  |           |  Q1,00  | PCWC |
|        TABLE ACCESS FULL| T3        |  Q1,00  | PCWP |
|     PX RECEIVE           |           |  Q1,02  | PCWP |
|      PX SEND HASH        | :TQ10001  |  Q1,01  | P->P |
|       PX BLOCK ITERATOR  |           |  Q1,01  | PCWC |
|        TABLE ACCESS FULL| T2        |  Q1,01  | PCWP |
-----------------------------------------------------------
```

```
-----------------------------------------------------------
| Id  | Operation                | Name      |    TQ   |IN-OUT|
-----------------------------------------------------------
|   0 | SELECT STATEMENT         |           |         |      |
|   1 |  PX COORDINATOR          |           |         |      |
|   2 |   PX SEND QC (RANDOM)    | :TQ10001  |  Q1,01  | P->S |
|*  3 |    HASH JOIN             |           |  Q1,01  | PCWP |
|   4 |     PX RECEIVE           |           |  Q1,01  | PCWP |
|   5 |      PX SEND BROADCAST   | :TQ10000  |  Q1,00  | P->P |
|   6 |       PX BLOCK ITERATOR  |           |  Q1,00  | PCWC |
|   7 |        TABLE ACCESS FULL| T_1       |  Q1,00  | PCWP |
|   8 |     PX BLOCK ITERATOR    |           |  Q1,01  | PCWC |
|   9 |      TABLE ACCESS FULL   | T_2       |  Q1,01  | PCWP |
-----------------------------------------------------------
```

Ein Seiteneffekt der BROADCAST-Verteilung ist, dass eben nur eine der beiden Datenquellen umverteilt werden muss, und dadurch die oben erwähnte Zwischenpufferung der Daten (HASH JOIN BUFFERED) nicht notwendig ist, da bei dem gezeigten Ausführungsplan nicht mehr als eine Umverteilung der Daten gleichzeitig aktiv ist.

Oracle 12c führt eine neue Verteilungsmethode PX SEND HYBRID HASH ein, die zwei Aufgaben erfüllt: Erstens kann sie dynamisch zur Laufzeit zwischen BROADCAST und HASH-Verteilung umschalten, und zweitens kann sie – leider derzeit nur in einer beschränkten Anzahl von Szenarien – eine Ungleichverteilung der HASH-Verteilung verursacht durch populäre Werte erkennen und korrigieren (neuer [NO_]PQ_SKEW Hint).

```
-------------------------------------------------------------------------
| Id  | Operation                  | Name     |    TQ  |IN-OUT|
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |          |        |      |
|   1 |  PX COORDINATOR            |          |        |      |
|   2 |   PX SEND QC (RANDOM)      | :TQ10002 |  Q1,02 | P->S |
|*  3 |    HASH JOIN BUFFERED      |          |  Q1,02 | PCWP |
|   4 |     PX RECEIVE             |          |  Q1,02 | PCWP |
|   5 |      PX SEND HYBRID HASH   | :TQ10000 |  Q1,00 | P->P |
|   6 |       STATISTICS COLLECTOR |          |  Q1,00 | PCWC |
|   7 |        PX BLOCK ITERATOR   |          |  Q1,00 | PCWC |
|   8 |         TABLE ACCESS FULL  | T_1      |  Q1,00 | PCWP |
|   9 |     PX RECEIVE             |          |  Q1,02 | PCWP |
|  10 |      PX SEND HYBRID HASH   | :TQ10001 |  Q1,01 | P->P |
|  11 |       PX BLOCK ITERATOR    |          |  Q1,01 | PCWC |
|  12 |        TABLE ACCESS FULL   | T_2      |  Q1,01 | PCWP |
-------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------
| Id  | Operation                  | Name     |    TQ  |IN-OUT|
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |          |        |      |
|   1 |  PX COORDINATOR            |          |        |      |
|   2 |   PX SEND QC (RANDOM)      | :TQ10002 |  Q1,02 | P->S |
|*  3 |    HASH JOIN BUFFERED      |          |  Q1,02 | PCWP |
|   4 |     PX RECEIVE             |          |  Q1,02 | PCWP |
|   5 |      PX SEND HYBRID HASH   | :TQ10000 |  Q1,00 | P->P |
|   6 |       STATISTICS COLLECTOR |          |  Q1,00 | PCWC |
|   7 |        PX BLOCK ITERATOR   |          |  Q1,00 | PCWC |
|   8 |         TABLE ACCESS FULL  | T_1      |  Q1,00 | PCWP |
|   9 |     PX RECEIVE             |          |  Q1,02 | PCWP |
|  10 |      PX SEND HYBRID HASH (SKEW)| :TQ10001 |  Q1,01 | P->P |
|  11 |       PX BLOCK ITERATOR    |          |  Q1,01 | PCWC |
|  12 |        TABLE ACCESS FULL   | T_2      |  Q1,01 | PCWP |
-------------------------------------------------------------------------
```

Beide erweiterten Möglichkeiten der HYBRID HASH Verteilungsmöglichkeiten sind in der aktuellen Version leider deutlich vom Nutzen her eingeschränkt. Die dynamische Entscheidung zwischen BROADCAST und HASH verwendet derzeit einen extrem niedrigen Schwellenwert (2*DOP Zeilen, also bei einem Parallelitätsgrad von 8 sind das 16 Zeilen), so dass in der überwiegenden Anzahl der Fälle eh die HASH Verteilung ausgewählt wird, und darüber hinaus kann man an dem Ausführungsplan oben erkennen, dass der Vorteil der BROADCAST-Verteilung, dass nur eine der beiden Datenquellen umverteilt werden muss, selbst bei dynamischer Auswahl der BROADCAST-Verteilung nicht gegeben ist – die zweite Datenquelle wird in diesem Fall nämlich per ROUND-ROBIN-Verfahren umverteilt, daher ist auch die BUFFERED-Variante des HASH JOIN erforderlich.

In zukünftigen Versionen soll dies verbessert werden. Die automatische Erkennung von Ungleichverteilung (SKEW) ist im Grunde eine sehr nützliche Erweiterung, nur leider kommt sie in

derzeitigen Implementierung noch mit zu vielen Einschränkungen, so dass es immer noch eine Vielzahl von Szenarien gibt, die nicht abgedeckt sind.

Concurrent UNION ALL

12c führt ein neues Feature ein, dass die gleichzeitige Ausführung mehrerer Unteroperationen eines UNION ALL erlaubt. Dieses Feature ist im Grunde nur relevant, wenn das UNION ALL aus einer Mixtur von parallelen und seriellen Operationen besteht, wobei in Ausnahmefällen es auch einen Unterschied machen kann, wenn das UNION ALL nur aus parallelen Operationen besteht, dann nämlich, wenn es zu ungleicher Verteilung der Arbeit innerhalb einer parallelen Unteroperation kommt.

Vor 12c werden die einzelnen Teile eines UNION ALL sequentiell nacheinander ausgeführt, und die seriellen Teile des UNION ALL werden vom Query Coordinator selbst ausgeführt. Die Daten werden dann auf die Parallel Execution Server verteilt – dabei tritt ein weiteres Phänomen auf: Wenn der Query Coordinator selbst beteiligt und eine Datenverteilung aktiv ist, werden weitere, mögliche Parallelverarbeitungen temporär verhindert mittels einer zusätzlichen BUFFER SORT Operation. Es scheint eine Regel zu geben, dass bei Aktivität des Query Coordinators keine gleichzeitige Parallelverarbeitung möglich ist.

```
-------------------------------------------------------------------------------
| Id  | Operation                  | Name     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |          |        |      |            |
|   1 |  SORT AGGREGATE            |          |        |      |            |
|   2 |   PX COORDINATOR           |          |        |      |            |
|   3 |    PX SEND QC (RANDOM)     | :TQ10008 |  Q1,08 | P->S | QC (RAND)  |
|   4 |     SORT AGGREGATE         |          |  Q1,08 | PCWP |            |
|   5 |      VIEW                  |          |  Q1,08 | PCWP |            |
|   6 |       UNION-ALL            |          |  Q1,08 | PCWP |            |
|   7 |        BUFFER SORT         |          |  Q1,08 | PCWC |            |
|   8 |         PX RECEIVE         |          |  Q1,08 | PCWP |            |
|   9 |          PX SEND ROUND-ROBIN | :TQ10000 |      | S->P | RND-ROBIN  |
|* 10 |           TABLE ACCESS FULL | T2       |        |      |            |
|  11 |        BUFFER SORT         |          |  Q1,08 | PCWC |            |
|  12 |         PX RECEIVE         |          |  Q1,08 | PCWP |            |
|  13 |          PX SEND ROUND-ROBIN| :TQ10001 |        | S->P | RND-ROBIN  |
|* 14 |           TABLE ACCESS FULL | T2       |        |      |            |
|  15 |        BUFFER SORT         |          |  Q1,08 | PCWC |            |
|  16 |         PX RECEIVE         |          |  Q1,08 | PCWP |            |
|  17 |          PX SEND ROUND-ROBIN| :TQ10002 |        | S->P | RND-ROBIN  |
|* 18 |           TABLE ACCESS FULL | T2       |        |      |            |
|  19 |        PX BLOCK ITERATOR   |          |  Q1,08 | PCWC |            |
|* 20 |         TABLE ACCESS FULL  | T_2      |  Q1,08 | PCWP |            |
|  21 |        BUFFER SORT         |          |  Q1,08 | PCWC |            |
|  22 |         PX RECEIVE         |          |  Q1,08 | PCWP |            |
|  23 |          PX SEND ROUND-ROBIN| :TQ10003 |        | S->P | RND-ROBIN  |
|* 24 |           TABLE ACCESS FULL | T2       |        |      |            |
-------------------------------------------------------------------------------
```

Mit 12c werden die vorhandenen Parallel Execution Server mittels eines neuen Operator PX SELECTOR auf die Operationen verteilt (neuer [NO_]PQ_CONCURRENT_UNION Hint). Insbesondere werden die seriellen Ausführungsteile jetzt von jeweils einem mittels PX SELECTOR zugeordnetem Parallel Execution Server ausgeführt, und nicht mehr vom Query Coordinator selbst. Dadurch entfällt der zusätzliche BUFFER SORT, als auch die Notwendigkeit der Datenverteilung auf

die Parallel Execution Server. Der neue PX SELECTOR Operator kann auch unabhängig von UNION ALL zum Einsatz kommen, wenn in 12c serielle Teile eines parallelen Ausführungsplans existieren. Allerdings wird der neue Operator nicht in allen Fällen eingesetzt, was heisst, dass es auch in 12c immer noch zu seriellen Ausführungen des Query Coordinators kommen kann, mit den beschriebenen Seiteneffekten der zusätzlichen BUFFER SORT Operationen.

1 Slave Verteilungsmethode

In Bezug auf die Aufteilung eines parallelen Ausführungsplans auf mehrere DFO Trees führt Oracle 12c eine neue Verteilungsmethode ein, die, wenn sie zum Einsatz kommt, eine Aufteilung nicht mehr notwendig macht.

Bei Einsatz oder Kombination bestimmter SQL Features, wie zum Beispiel bestimmter analytischer Funktionen wie zum Beispiel LAG oder LEAD, muss Oracle unter Umständen einen Teil der Daten in einem nicht-parallelen Teil des Ausführungsplans verarbeiten. Dabei kam es in der Vergangenheit zur Aufteilung des Ausführungsplans in mehrere DFO Trees.

```
-------------------------------------------------------------------------------------
| Id  | Operation                          | Name     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |          |        |      |            |
|   1 |  COUNT                             |          |        |      |            |
|*  2 |   FILTER                           |          |        |      |            |
|   3 |    PX COORDINATOR                  |          |        |      |            |
|   4 |     PX SEND QC (RANDOM)            | :TQ30002 | Q3,02  | P->S | QC (RAND)  |
|   5 |      VIEW                          |          | Q3,02  | PCWP |            |
|*  6 |       HASH JOIN                    |          | Q3,02  | PCWP |            |
|   7 |        PX RECEIVE                  |          | Q3,02  | PCWP |            |
|   8 |         PX SEND HASH               | :TQ30001 | Q3,01  | P->P | HASH       |
|   9 |          PX BLOCK ITERATOR         |          | Q3,01  | PCWC |            |
|  10 |           TABLE ACCESS FULL        | T6       | Q3,01  | PCWP |            |
|  11 |        BUFFER SORT                 |          | Q3,02  | PCWC |            |
|  12 |         PX RECEIVE                 |          | Q3,02  | PCWP |            |
|  13 |          PX SEND HASH              | :TQ30000 |        | S->P | HASH       |
|* 14 |           HASH JOIN                |          |        |      |            |
|  15 |            VIEW                    |          |        |      |            |
|  16 |             COUNT                  |          |        |      |            |
|  17 |              PX COORDINATOR        |          |        |      |            |
|  18 |               PX SEND QC (RANDOM)  | :TQ10000 | Q1,00  | P->S | QC (RAND)  |
|  19 |                PX BLOCK ITERATOR   |          | Q1,00  | PCWC |            |
|  20 |                 TABLE ACCESS FULL  | T2       | Q1,00  | PCWP |            |
|  21 |            VIEW                    |          |        |      |            |
|  22 |             COUNT                  |          |        |      |            |
|  23 |              PX COORDINATOR        |          |        |      |            |
|  24 |               PX SEND QC (RANDOM)  | :TQ20000 | Q2,00  | P->S | QC (RAND)  |
|  25 |                PX BLOCK ITERATOR   |          | Q2,00  | PCWC |            |
|  26 |                 TABLE ACCESS FULL  | T4       | Q2,00  | PCWP |            |
-------------------------------------------------------------------------------------
```

In 12c werden manche solcher Ausführungspläne nun mittels der neuen PX 1 SLAVE-Verteilungsmethode in einem DFO Tree zusammengefasst:

```
-------------------------------------------------------------------------------------
| Id  | Operation                          | Name     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |          |        |      |            |
|   1 |  COUNT                             |          |        |      |            |
|*  2 |   FILTER                           |          |        |      |            |
|   3 |    PX COORDINATOR                  |          |        |      |            |
|   4 |     PX SEND QC (RANDOM)            | :TQ10004 |  Q1,04 | P->S | QC (RAND)  |
|   5 |      VIEW                          |          |  Q1,04 | PCWP |            |
|*  6 |       HASH JOIN BUFFERED           |          |  Q1,04 | PCWP |            |
|   7 |        PX RECEIVE                  |          |  Q1,04 | PCWP |            |
|   8 |         PX SEND HASH               | :TQ10002 |  Q1,02 | P->P | HASH       |
|   9 |          PX BLOCK ITERATOR         |          |  Q1,02 | PCWC |            |
|  10 |           TABLE ACCESS FULL        | T6       |  Q1,02 | PCWP |            |
|  11 |        PX RECEIVE                  |          |  Q1,04 | PCWP |            |
|  12 |         PX SEND HASH               | :TQ10003 |  Q1,03 | S->P | HASH       |
|* 13 |          HASH JOIN BUFFERED        |          |  Q1,03 | SCWC |            |
|  14 |           VIEW                     |          |  Q1,03 | SCWC |            |
|  15 |            COUNT                   |          |  Q1,03 | SCWP |            |
|  16 |             PX RECEIVE             |          |  Q1,03 | SCWP |            |
|  17 |              PX SEND 1 SLAVE       | :TQ10000 |  Q1,00 | P->S | 1 SLAVE    |
|  18 |               PX BLOCK ITERATOR    |          |  Q1,00 | PCWC |            |
|  19 |                TABLE ACCESS FULL   | T2       |  Q1,00 | PCWP |            |
|  20 |           VIEW                     |          |  Q1,03 | SCWC |            |
|  21 |            COUNT                   |          |  Q1,03 | SCWP |            |
|  22 |             PX RECEIVE             |          |  Q1,03 | SCWP |            |
|  23 |              PX SEND 1 SLAVE       | :TQ10001 |  Q1,01 | P->S | 1 SLAVE    |
|  24 |               PX BLOCK ITERATOR    |          |  Q1,01 | PCWC |            |
|  25 |                TABLE ACCESS FULL   | T4       |  Q1,01 | PCWP |            |
-------------------------------------------------------------------------------------
```

Die Ausführung der entsprechenden Teile ist immer noch seriell, so dass die Parallelisierung sich nicht von den früheren Ausführungsplänen unterscheidet, allerdings gibt es nicht mehr die oben beschriebenen Seiteneffekte mit mehr Parallel Execution Servern als erwartet, unterschiedliche Parallelitätsgrade pro DFO Tree und Overhead durch Allokation und Freigabe von Parallel Execution Servern. Ein möglicher Nachteil der neuen Verteilungsmethode kann in dem obigen Ausführungsplan gesehen werden: Die beiden HASH JOINs werden nun in der BUFFERED-Variante ausgeführt ausgelöst durch die entsprechende Limitierung, dass nur eine Verteilung pro DFO Tree gleichzeitig aktiv sein kann.

Parallel Filter

Beinhaltete in Versionen vor Oracle 12c ein paralleler Ausführungsplan einen FILTER Operator mit Unterabfragen, mussten die Daten immer erst seriell im Query Coordinator verarbeitet werden, bevor die Unterabfragen ausgeführt werden konnten:

```
-------------------------------------------------------------------------------
| Id  | Operation                | Name     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |          |        |      |            |
|   1 |  SORT AGGREGATE          |          |        |      |            |
|*  2 |   FILTER                 |          |        |      |            |
|   3 |    PX COORDINATOR        |          |        |      |            |
|   4 |     PX SEND QC (RANDOM)  | :TQ20000 |  Q2,00 | P->S | QC (RAND)  |
|   5 |      PX BLOCK ITERATOR   |          |  Q2,00 | PCWC |            |
|   6 |       TABLE ACCESS FULL  | T_1      |  Q2,00 | PCWP |            |
|   7 |    PX COORDINATOR        |          |        |      |            |
|   8 |     PX SEND QC (RANDOM)  | :TQ10000 |  Q1,00 | P->S | QC (RAND)  |
|   9 |      PX BLOCK ITERATOR   |          |  Q1,00 | PCWC |            |
|* 10 |       TABLE ACCESS FULL  | T_1      |  Q1,00 | PCWP |            |
-------------------------------------------------------------------------------
```

Ein weiterer Nachteil des seriellen Filters war die Möglichkeit, wie oben im Ausführungsplan zu sehen, dass wieder mehrere DFO Trees entstehen können. In diesem Fall wird ein DFO Tree sogar potentiell vielfach ausgeführt, so dass die beschriebenen Nachteile der Allokation und Freigabe der entsprechenden Parallel Execution Server hier relevant sein können.

In Oracle 12c kann der FILTER Operator nun in den Parallel Execution Servern ausgeführt werden, was große Performance-Vorteile bieten kann, wenn viel Zeit in den Filter-Unterabfragen verbracht wird.

```
-------------------------------------------------------------------------------
| Id  | Operation                | Name     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |          |        |      |            |
|   1 |  SORT AGGREGATE          |          |        |      |            |
|   2 |   PX COORDINATOR         |          |        |      |            |
|   3 |    PX SEND QC (RANDOM)   | :TQ10000 |  Q1,00 | P->S | QC (RAND)  |
|   4 |     SORT AGGREGATE       |          |  Q1,00 | PCWP |            |
|*  5 |      FILTER              |          |  Q1,00 | PCWC |            |
|   6 |       PX BLOCK ITERATOR  |          |  Q1,00 | PCWC |            |
|   7 |        TABLE ACCESS FULL | T_1      |  Q1,00 | PCWP |            |
|*  8 |      TABLE ACCESS FULL   | T_1      |        |      |            |
-------------------------------------------------------------------------------
```

Je nach Daten und Ausführungsplan können hier für den FILTER noch zusätzliche Verteilungsmethoden verwendet werden (neuer PQ_FILTER Hint), im obigen Beispiel kommt aber keine zusätzliche Verteilung zum Einsatz.

Was etwas irritierend ist, dass die Ausführung der Unterabfragen in den Parallel Execution Servern stattfindet (Im Plan Operation ID = 8), die Operation aber nicht parallel markiert ist (TQ / IN-OUT Spalte). Der Plan suggeriert also eine serielle Ausführung, zur Laufzeit wird der TABLE ACCESS FULL T1 aber in jedem Parallel Execution Server separat pro Iteration ausgeführt.

Weitere neue Features

PQ_REPLICATE – wiederholte Ausführung eines Zugriffs in den Parallel Execution Servern anstatt BROADCAST-Verteilung

Wird ein paralleler Full Table Scan ausgeführt, der anschließend per BROADCAST-Verteilung an die Parallel Execution Server dupliziert wird, kann Oracle 12c eine alternative Form der Ausführung verwenden. Bisher sah ein solcher Ausführungsplan zum Beispiel so aus:

```
-------------------------------------------------------------------------------
| Id  | Operation               | Name     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |          |        |      |            |
|   1 |  PX COORDINATOR         |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)   | :TQ10001 |  Q1,01 | P->S | QC (RAND)  |
|*  3 |    HASH JOIN            |          |  Q1,01 | PCWP |            |
|   4 |     PX RECEIVE          |          |  Q1,01 | PCWP |            |
|   5 |      PX SEND BROADCAST  | :TQ10000 |  Q1,00 | P->P | BROADCAST  |
|   6 |       PX BLOCK ITERATOR |          |  Q1,00 | PCWC |            |
|   7 |        TABLE ACCESS FULL| T1       |  Q1,00 | PCWP |            |
|   8 |     PX BLOCK ITERATOR   |          |  Q1,01 | PCWC |            |
|   9 |      TABLE ACCESS FULL  | T2       |  Q1,01 | PCWP |            |
-------------------------------------------------------------------------------
```

Hier wird auf T1 per parallelem Full Table Scan zugegriffen (PX BLOCK ITERATOR zeigt an, dass jeder Parallel Execution Server zugewiesene Chunks von T1 scannt, also jeder Parallel Execution Server nur einen Teil des Segments verarbeitet), und dann mittels BROADCAST an jeden empfangenden Parallel Execution Server des anderen Sets dupliziert.

In 12c kann dieser Plan stattdessen so aussehen:

```
-------------------------------------------------------------------------------
| Id  | Operation               | Name     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |          |        |      |            |
|   1 |  PX COORDINATOR         |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)   | :TQ10000 |  Q1,00 | P->S | QC (RAND)  |
|*  3 |    HASH JOIN            |          |  Q1,00 | PCWP |            |
|   4 |     TABLE ACCESS FULL   | T1       |  Q1,00 | PCWP |            |
|   5 |     PX BLOCK ITERATOR   |          |  Q1,00 | PCWC |            |
|   6 |      TABLE ACCESS FULL  | T2       |  Q1,00 | PCWP |            |
-------------------------------------------------------------------------------
```

Hier wird der Full Table Scan von T1 jetzt vollständig in jedem Parallel Execution Server verarbeitet (also keine Teilverarbeitung mittels Chunking mehr, da kein PX BLOCK ITERATOR), so dass jeder Parallel Execution Server T1 vollständig liest. Daher müssen die Daten auch nicht mehr mittels BROADCAST zwischen zwei Sets versendet und dabei dupliziert werden – das Endresultat ist das gleiche: Jeder Parallel Execution Server hat eine vollständige Kopie von T1 verarbeitet. Der obige Ausführungsplan kommt daher auch mit einem Set von Parallel Execution Servern aus, was aber je nach restlichen Operationen nicht der Fall sein muss – falls andere Operationen innerhalb des gleichen DFO Trees Datenverteilungen benötigen, werden zwei Sets allokiert werden.

Neuer Operator EXPRESSION EVALUATION für skalare Unterabfragen bei Parallelverarbeitung

Kommen skalare Unterabfragen in parallelen Ausführungsplänen zum Einsatz, kann Oracle 12c einen neuen Operator EXPRESSION EVALUATION verwenden.

Bisher sah ein solcher Ausführungsplan zum Beispiel so aus:

```
-------------------------------------------------------------------------------
| Id  | Operation                | Name     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |          |        |      |            |
|   1 |  PX COORDINATOR          |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)    | :TQ10000 |  Q1,00 | P->S | QC (RAND)  |
|   3 |    PX BLOCK ITERATOR     |          |  Q1,00 | PCWC |            |
|*  4 |     TABLE ACCESS FULL    | T2       |  Q1,00 | PCWP |            |
|   5 |  PX COORDINATOR          |          |        |      |            |
|   6 |   PX SEND QC (RANDOM)    | :TQ20000 |  Q2,00 | P->S | QC (RAND)  |
|   7 |    PX BLOCK ITERATOR     |          |  Q2,00 | PCWC |            |
|*  8 |     TABLE ACCESS FULL    | T2       |  Q2,00 | PCWP |            |
|   9 |  PX COORDINATOR          |          |        |      |            |
|  10 |   PX SEND QC (RANDOM)    | :TQ30001 |  Q3,01 | P->S | QC (RAND)  |
|* 11 |    HASH JOIN             |          |  Q3,01 | PCWP |            |
|  12 |     PX RECEIVE           |          |  Q3,01 | PCWP |            |
|  13 |      PX SEND BROADCAST   | :TQ30000 |  Q3,00 | P->P | BROADCAST  |
|  14 |       PX BLOCK ITERATOR  |          |  Q3,00 | PCWC |            |
|* 15 |        TABLE ACCESS FULL | T2       |  Q3,00 | PCWP |            |
|  16 |     PX BLOCK ITERATOR    |          |  Q3,01 | PCWC |            |
|* 17 |      TABLE ACCESS FULL   | T2       |  Q3,01 | PCWP |            |
-------------------------------------------------------------------------------
```

Eine der Merkwürdigkeiten mit skalaren Unterabfragen ist, dass sie den üblichen Regeln widersprechen, die die Reihenfolge, in der die Operatoren eines Ausführungsplans ausgeführt werden, festlegen – was heisst, dass im obigen Ausführungsplan die Ausnahme besteht, dass die Hauptabfrage erst bei Operation 9 beginnt,. Die Operatoren 1-4 und 5-8 stellen zwei skalare Unterabfragen dar, die grundsätzlich pro generierter Zeile der Hauptfrage ausgeführt werden (Scalar Subquery Caching kann diese Anzahl reduzieren, muss aber nicht). Gemäß den allgemeinen Regeln würde der Ausführungsplan eigentlich bei Operation 1 bzw. 4 beginnen Man kann hier auch sehen, dass bei paralleler Ausführung mit skalaren Unterabfragen mehrere DFO Trees entstehen können, was gerade im Falle von skalaren Unterabfragen eine schlechte Idee ist, da sie ja potentiell sehr häufig ausgeführt werden können, und damit wieder der Overhead der Allokation / Freigabe von Parallel Execution Servern eine signifikante Rolle spielen kann.

In Oracle 12c kann dieser Plan so aussehen:

```
-------------------------------------------------------------------------------
| Id  | Operation                  | Name     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |          |        |      |            |
|   1 |  PX COORDINATOR            |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)      | :TQ10002 |  Q1,02 | P->S | QC (RAND)  |
|   3 |    EXPRESSION EVALUATION   |          |  Q1,02 | PCWC |            |
|*  4 |     HASH JOIN BUFFERED     |          |  Q1,02 | PCWP |            |
|   5 |      PX RECEIVE            |          |  Q1,02 | PCWP |            |
|   6 |       PX SEND HYBRID HASH  | :TQ10000 |  Q1,00 | P->P | HYBRID HASH|
|   7 |        STATISTICS COLLECTOR|          |  Q1,00 | PCWC |            |
|   8 |         PX BLOCK ITERATOR  |          |  Q1,00 | PCWC |            |
|*  9 |          TABLE ACCESS FULL | T2       |  Q1,00 | PCWP |            |
|  10 |      PX RECEIVE            |          |  Q1,02 | PCWP |            |
|  11 |       PX SEND HYBRID HASH  | :TQ10001 |  Q1,01 | P->P | HYBRID HASH|
|  12 |        PX BLOCK ITERATOR   |          |  Q1,01 | PCWC |            |
|* 13 |         TABLE ACCESS FULL  | T2       |  Q1,01 | PCWP |            |
|* 14 |    TABLE ACCESS FULL       | T2       |        |      |            |
|* 15 |    TABLE ACCESS FULL       | T2       |        |      |            |
-------------------------------------------------------------------------------
```

Hier kommt der neue EXPRESSION EVALUATION Operator zum Einsatz, der zumindest das Problem der nicht den allgemeinen Regeln entsprechenden Ausführungsreihenfolge des Plans korrigiert, da jetzt die skalaren Unterabfragen unterhalb der Hauptabfrage als Kind-Elemente des EXPRESSION EVALUATION-Operators stehen.

Ähnlich wie beim neuen parallelen Filter-Operator (siehe oben) ist es verwirrend, dass die skalaren Unterabfragen nicht parallel markiert sind, der Plan also suggeriert, die Unterabfragen würden seriell ausgeführt werden. Zur Laufzeit werden die beiden Full Table Scans aber in den Parallel Execution Servern ausgeführt, also jeder Aufruf der skalaren Unterabfrage führt einen kompletten Full Table Scan innerhalb des Parallel Execution Servers aus.

In der derzeit aktuellen Version 12.1.0.2 verhält sich der EXPRESSION EVALUATION Operator zur Laufzeit merkwürdig, in dem er dazu führt, dass die skalaren Unterabfragen anscheinend häufiger als notwendig ausgeführt werden, wie sich in meinen Tests gezeigt hat. Das Phänomen ist noch nicht abschließend untersucht, daher kann hier noch keine definitive Aussage über das Verhalten gemacht werden.

Zusammenfassung

Mit der Version 12c führt Oracle so viele neuen Operatoren für parallele Verarbeitung ein, wie schon seit vielen Jahren nicht mehr. Auch wenn die Neuerungen insgesamt nicht so revolutionär sind wie zum Beispiel die neue InMemory Option, ist es erfreulich zu sehen, dass auch auf diesem Gebiet Innovationen nicht ausbleiben. Es ist zu erwarten, dass mit Version 12.2 weitere Neuerungen auch in diesem Bereich kommen, beziehungsweise hier beschriebene, neue Features weiter verbessert werden, wie zum Beispiel die neue HYBRID HASH-Verteilungsmethode.

Kontaktadresse:
Randolf Geist
Unabhängiger Berater
Lilienstraße 37
68535 Edingen-Neckarhausen

Telefon: +49 (0) 170-758 1171
E-Mail randolf.geist@oracle-performance.de
Internet: http://www.oracle-performance.de