Parallel Processing
The difference between 10g and 11gR2

Jean-Pierre Dijcks
Oracle
Redwood City, CA, USA

Keywords:

Automatic DOP, data warehouse, database, parallel execution

Introduction

This paper focuses on the improvements and enhancements to parallel execution when compared to the world of Oracle Database 10g.

Problem Statement

The basic goal of parallel execution in Oracle Database 11g Release 2 is to solve a managaeability challenge always coming up in the realm of parallel execution. This problem being: how do I get parallelism out of the box for the queries and actions that require parallelism.

That is directly the largest difference between a database in the 10g timeframe and a database in the 11g Release 2 timeframe. When running a database in automatic degree of parallelism mode a job will by default run in parallel when that is appropriate.

Apart from that problem statement a number of other problems are tackled:

1. Overloading a machine with too many processes trying to run at the same time
2. Drastic changes in elapsed time when a statement that usually runs in parallel goes serial due to lack of resources
3. Adaptive parallelism gives unpredictable run times due to capping of parallelism based on the number of sessions present
4. Leveraging memory resident data for parallel processing was not possible (not covered in this paper!)

The following will describe how to leverage the new functionality and make the 10g problems disappear.

Avoid using adaptive parallelism

In 10g, adaptive parallelism looked at the individual SQL statement to determine the ideal DOP for a statement. The old adaptive parallelism capabilities were assessing the parallel resources given to an individual statement based on the actual system workload: Oracle decided at SQL execution time whether a parallel operation should receive the requested DOP or be throttled down to a lower DOP based on the concurrent workload on the system.

In a system that makes aggressive use of parallel execution by using a high DOP the adaptive algorithm would throttle down the DOP with only few operations running in parallel. While the algorithm will still ensure optimal resource utilization, users may experience inconsistent response times. Using solely the adaptive parallelism capabilities in an environment that requires deterministic response times is not advised.

Adaptive parallelism is controlled through the database initialization parameter:

```
PARALLEL_ADAPTIVE_MULTI_USER = true | false
```

Auto DOP is the preferred method over adaptive parallelism to control the DOP in a multi-user environment and this is one of the first and most important changes to make when moving to an 11g Release 2 environment.

The idea behind calculating the Automatic Degree of Parallelism is to find the highest possible DOP (ideal DOP) that still scales. In other words, if we were to increase the DOP even more above a certain DOP we would see a tailing off of the performance curve and the resource cost / performance would become less optimal. Therefore the ideal DOP is the best resource/performance point for that statement.

Avoiding overloading a machine or dramatic decreases in performance

This section discusses concurrency and explains some of the relevant parameters and their impact, specifically in a mixed workload environment. Any mixed workload environment will have statements with varying degrees of parallelism running. All of it needs to live within its means of the resources available on the entire system.

The goal of Queuing

On a normal production system we should see statements running concurrently. On a Database Machine we typically see high concurrency rates, so we need to find a way to deal with both high DOP's and high concurrency.

Queuing is intended to make sure we:

1. Don't throttle down a DOP because other statements are running on the system
2. Stay within the physical limits of a system's processing power

Instead of making statements go at a lower DOP we queue them to make sure they will get all the resources they want to run efficiently without trashing the system. The theory - and hopefully - practice is that by giving a statement the optimal DOP the sum of all statements runs faster with queuing than without queuing.

Increasing the Number of Potential Parallel Statements

To determine how many statements we will consider running in parallel a single parameter should be looked at. That parameter is called PARALLEL_MIN_TIME_THRESHOLD. The default value is set to 10 seconds.

Now, if you have a system where you have two groups of queries, serial short running and potentially parallel long running ones, you may want to worry only about the long running ones with this parallel statement threshold. As an example, lets assume the short running stuff runs on average between 1 and 15 seconds in serial (and the business is quite happy with that). The long running stuff is in the realm of 1 - 5 minutes.

It might be a good choice to set the threshold to somewhere north of 30 seconds. That way the short running queries all run serial as they do today (if it is not broken, don't fix it) and allows the long running ones to be evaluated for (higher degrees of) parallelism. This makes sense because the longer running ones are (at least in theory) more interesting to unleash a parallel processing model on and the benefits of running these in parallel are much more significant (again, that is mostly the case).

Setting a System Wide Maximum DOP for a Statement
Now that you know how to control how many of your statements are considered to run in parallel, let’s talk about the specific degree of any given statement that will be evaluated. As the fundamentals paper describes this is controlled by PARALLEL_DEGREE_LIMIT. This parameter controls the degree on the entire cluster and by default it is CPU (meaning it equals Default DOP).

For the sake of an example, let's say our Default DOP is 32. Looking at our 5 minute queries from the previous paragraph, the limit to 32 means that none of the statements that are evaluated for Auto DOP ever runs at more than DOP of 32.

Concurrently Running a High DOP
A basic assumption about running high DOP statements at high concurrency is that you at some point in time (and this is true on any parallel processing platform!) will run into a resource limitation. And yes, you can then buy more hardware (e.g. expand the Database Machine in Oracle's case), but that is not the point of this section...
The goal is to find a balance between the highest possible DOP for each statement and the number of statements running concurrently, but with an emphasis on running each statement at that highest efficiency DOP.

The PARALLEL_SERVER_TARGET parameter is the all important concurrency slider here. Setting this parameter to a higher number means more statements get to run at their maximum parallel degree before queuing kicks in. PARALLEL_SERVER_TARGET is set per instance (so needs to be set to the same value on all 8 nodes in a full rack Database Machine). Just as a side note, this parameter is set in processes, not in DOP, which equates to 4* Default DOP (2 processes for a DOP, default value is 2 * Default DOP, hence a default of 4 * Default DOP).
Let's say we have PARALLEL_SERVER_TARGET set to 128. With our limit set to 32 (the default) we are able to run 4 statements concurrently at the highest DOP possible on this system before we start queuing. If these 4 statements are running, any next statement will be queued.

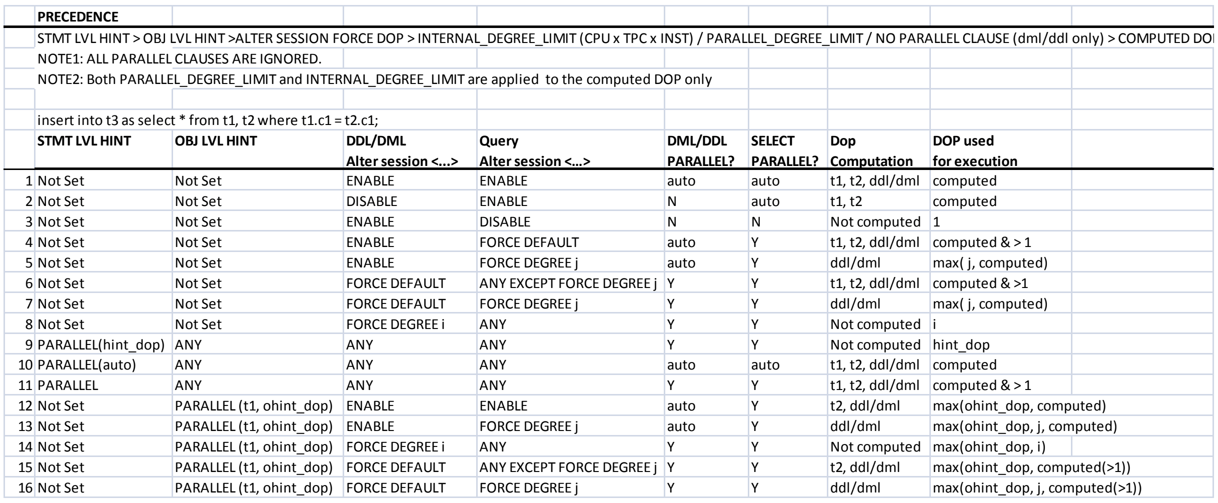
To run a system at high concurrency the PARALLEL_SERVER_TARGET should be raised from its default to be much closer (start with 60% or so) to PARALLEL_MAX_SERVERS.

By using both PARALLEL_SERVER_TARGET and PARALLEL_DEGREE_LIMIT you can control easily how many statements run concurrently at good DOPs without excessive queuing. Because each workload is a little different, it makes sense to plan ahead and look at these parameters and set these based on your requirements.

Dealing with Precedence and Overwrites of Automatic functionality

The following table is a reflection of the precedence of hints, things like alter session enable parallel DML when using Auto DOP. It is also important to understand how the DML and query parts work together and how they influence each other.
All of the below is based on a simple statement:

```
insert into t3 as select * from t1, t2 where t1.c1 = t2.c1;
```



PRECEDENCE

STMT LVL HINT > OBJ LVL HINT >ALTER SESSION FORCE DOP > INTERNAL_DEGREE_LIMIT (CPU x TPC x INST) / PARALLEL_DEGREE_LIMIT / NO PARALLEL CLAUSE (dml/ddl only) > COMPUTED DO

NOTE1: ALL PARALLEL CLAUSES ARE IGNORED.

NOTE2: Both PARALLEL_DEGREE_LIMIT and INTERNAL_DEGREE_LIMIT are applied to the computed DOP only

insert into t3 as select * from t1, t2 where t1.c1 = t2.c1;

	STMT LVL HINT	OBJ LVL HINT	DDL/DML Alter session <...>	Query Alter session <...>	DML/DDL PARALLEL?	SELECT PARALLEL?	Dop Computation	DOP used for execution
1	Not Set	Not Set	ENABLE	ENABLE	auto	auto	t1, t2, ddl/dml	computed
2	Not Set	Not Set	DISABLE	ENABLE	N	auto	t1, t2	computed
3	Not Set	Not Set	ENABLE	DISABLE	N	N	Not computed	1
4	Not Set	Not Set	ENABLE	FORCE DEFAULT	auto	Y	t1, t2, ddl/dml	computed & > 1
5	Not Set	Not Set	ENABLE	FORCE DEGREE j	auto	Y	ddl/dml	max(j, computed)
6	Not Set	Not Set	FORCE DEFAULT	ANY EXCEPT FORCE DEGREE j	Y	Y	t1, t2, ddl/dml	computed & >1
7	Not Set	Not Set	FORCE DEFAULT	FORCE DEGREE j	Y	Y	ddl/dml	max(j, computed)
8	Not Set	Not Set	FORCE DEGREE i	ANY	Y	Y	Not computed	i
9	PARALLEL(hint_dop)	ANY	ANY	ANY	Y	Y	Not computed	hint_dop
10	PARALLEL(auto)	ANY	ANY	ANY	auto	auto	t1, t2, ddl/dml	computed
11	PARALLEL	ANY	ANY	ANY	Y	Y	t1, t2, ddl/dml	computed & > 1
12	Not Set	PARALLEL (t1, ohint_dop)	ENABLE	ENABLE	auto	Y	t2, ddl/dml	max(ohint_dop, computed)
13	Not Set	PARALLEL (t1, ohint_dop)	ENABLE	FORCE DEGREE j	auto	Y	ddl/dml	max(ohint_dop, j, computed)
14	Not Set	PARALLEL (t1, ohint_dop)	FORCE DEGREE i	ANY	Y	Y	Not computed	max(ohint_dop, i)
15	Not Set	PARALLEL (t1, ohint_dop)	FORCE DEFAULT	ANY EXCEPT FORCE DEGREE j	Y	Y	t2, ddl/dml	max(ohint_dop, computed(>1))
16	Not Set	PARALLEL (t1, ohint_dop)	FORCE DEFAULT	FORCE DEGREE j	Y	Y	ddl/dml	max(ohint_dop, j, computed(>1))

Some explanatory words based on the lines in the picture above:
Line 1 => The cleanest way to run PX statements, where Auto DOP gets to do its work and we will use the computed DOP of the statement
Line 4 => Because a FORCE parallel is used, we must ensure that the DOP > 1
Line 9 => The statement level hint over rides all other means and we run the statement with the DOP in the hint

A word on internal degree limit. This is NOT (repeat NOT) a parameter you can find, or set or find an underscore for. It is the built in limit to DOPs (set to default DOP or parallel_degree_limit = CPU). It is an internal boundary to ensure we do not blast through the upper bound of CPU power. Also note, for any non-compute degree, those boundaries and limits do not apply. That in itself is a reason to go look at and understand Auto DOP.

Getting Started with 11g Release 2

This document does **not** discuss:

- The general theory and the theory of deriving a more optimal setting of parallel_degree_limit, parallel_servers_target and or the DBRM equivalents (see also next bullet). For more on these topics see:
    - The basics of Auto DOP (here)
    - Understanding statement queuing and deriving the parameter settings (here)
- Setting up and Using Database Resource Manager for Parallel Statement Queuing. For more on that topic see:
    - A small example using DBRM for workload management (here)
    - Workload Management based on execution times (here)

Basic parameters and settings to look at before you start

The assumption of this document is that you are running on an Oracle Exadata Database Machine X2-2. The settings also assume you are running at least 11.2.0.2 RDBMS.
Unless otherwise noted, the parameter settings are **per instance**.

Parameter Settings

Parallel_max_servers = 192
Parallel_min_servers = 96
Parallel_threads_per_cpu = 1
Parallel_adaptive_multi_user = FALSE
Parallel_degree_policy = AUTO
Parallel_servers_target = 128 -- note: this is number of processes
Parallel_degree_limit = 16 --note: this is in DOP (typically delivers 2* DOP processes)

IO Calibrate Statistics (per system):

In 11.2.0.2 you should also check that IO Calibrate has been run. Best to simply do a:

```
SQL> select * from V$IO_CALIBRATION_STATUS;

STATUS        CALIBRATION_TIME
------------- -------------------------------------------------------------
---
READY         04-JAN-11 10.04.13.104 AM
```

You should see that your IO Calibrate is READY and therefore Auto DOP is ready.

In any case, if you did not run the IO Calibrate step you will get the following note in the explain plan:

```
Note
-----
   - automatic DOP: skipped because of IO calibrate statistics are missing
```

One more note on calibrate_io, if you do not have asynchronous IO enabled you will see:

```
ERROR at line 1:
ORA-56708: Could not find any datafiles with asynchronous i/o capability
ORA-06512: at "SYS.DBMS_RMIN", line 463
ORA-06512: at "SYS.DBMS_RESOURCE_MANAGER", line 1296
ORA-06512: at line 7
```

While this is changed in some fixes to the calibrate procedure, you should really consider switching asynchronous IO on for your data warehouse.

Changing the behavior of your queries/system
The above assumes that you are running at relatively low DOPs for individual queries. Here are some of the knobs to turn to change behavior. If the parameter is not mentioned in the above section you can assume the value is left at the default setting.
All settings are dynamic and can therefore be set at session level and at system level without restarting the database.
Parallel_servers_target
Increase value => more queries run before queuing starts
Decrease value => fewer queries run before queuing starts
Parallel_degree_limit
Increase value => individual queries can run with higher DOPs
Decrease value => individual queries are capped to run with lower DOP
Parallel_min_time_threshold (default = Auto = 10seconds)
The estimated time of a serial plan must be larger than this setting before a statement is a candidate for a parallel run:
Increase value => Fewer queries qualify to be evaluated for a parallel plan
Decrease value => More queries qualify to be evaluated for a parallel plan

Contact address:

Jean-Pierre Dijcks
500 Oracle Parkway, M/S 4op7
Redwood City, CA, 94065

Phone:	+1 650 607 5394
Email	jean-pierre.dijcks@oracle.com
Internet:	blogs.oracle.com/datawarehousing