

An Oracle White Paper
October 2010

Parallel Execution and Workload Management for an Operational Data Warehouse

Table of Content

Introduction	1
Managing Mixed Workloads in a Data Warehouse	2
Understanding Parallel Statement Queuing.....	9
Working Examples.....	19
Summary.....	33

Introduction

Before reading this document it is important that you understand Oracle's parallel processing fundamentals discussed on [our website](#).

The focus of the paper you are now reading is on applying the fundamentals to an Oracle Database 11g Release 2 (11.2.0.2) environment and creating a data warehouse that can deal with a mixed workload running in parallel.

To address these topics this paper covers a number of topics:

- A short definition of mixed workloads
- Getting ready for workload management
- Automatic Degree of Parallelism and Concurrency
- Parallel Statement Queuing for workload management
- Examples:
 - User based workload management
 - Dynamic workload management based on statement execution times

Managing Mixed Workloads in a Data Warehouse

There are many different terms for a mixed workload in use today, active data warehousing, operational data warehousing etc. All indicate something similar, namely a diverse workload running on a data warehouse system concurrently. Whether it is continuous data loads while end users are querying data, or whether it is short running, OLTP like activities (both queries and trickle data loads) mixed in with more classic ad-hoc data intensive query patterns, all of these are mixed workloads as we define it in this paper.

Within Oracle we assume that the workload is managed in a single database instance. There is no need (functionality wise) within Oracle to worry about continuous writes while end users are querying because supports full read consistency across transactions. This Oracle functionality is notably different from other data warehouse databases which typically do not support these levels of read consistency.

Continuous Improvements

Workload management - the understanding and management of a set of diverse workloads on a system - is really an ecosystem with many participants. It is ever-changing and therefore is one of these things in life that will always be in motion. As the workload changes, or the environment in which the workload runs, adjustments will be required to ensure everything runs smoothly.

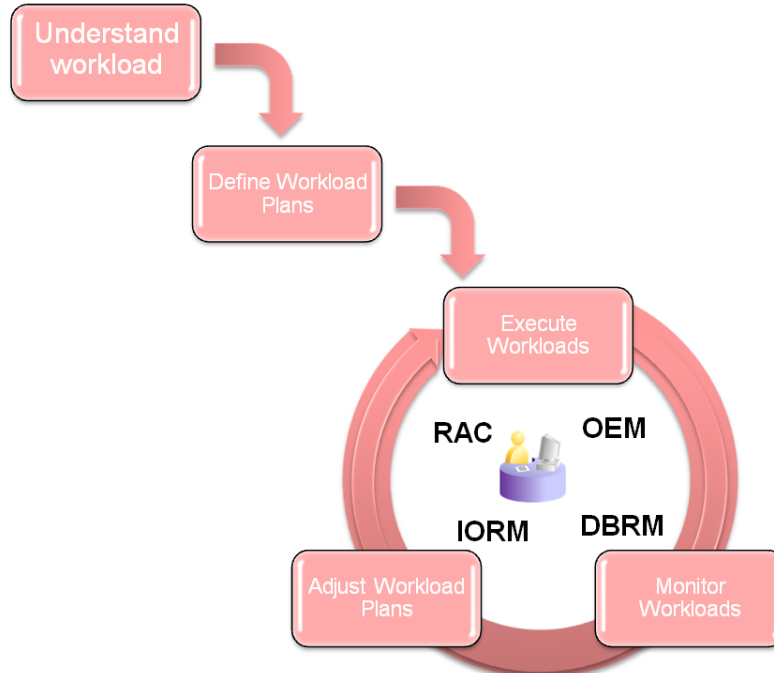


Figure 1. Continuous Improvements in Workload Management

At a high level, the cycle of continuous improvements begins with the definition of a workload plan. That definition should be based on a clear understanding of the actual workloads running on this system (more later on some of the required questions). That will be tested when the workloads are running on the system, and your main task is to monitor and adjust the workload. Adjusting - if all goes well and your plan is reasonable - is mostly required in the beginning when fine tuning of the plan is done.

Once the system stabilizes and all small exceptions to the plan are corrected your main task is to monitor. This whole cycle will repeat itself upon changes to the workloads or to the system. It is crucial that major changes are planned for and not just resolved in an adjustment.

Planning your Solution

To start creating effective workload management solutions it is crucial to understand the phases in above shown picture.

Understand the Workload

To understand the workload for your given system you will need to gather information on the following main points:

- Who is doing the work? - Which users are running workloads, which applications are running workloads?
- What types of work are done on the system? - Are these workloads batch, ad-hoc, resource intensive (which resources) and are these mixed or separated in some form?
- When are certain types being done? - Are there different workloads during different times of the day, are there different priorities during different time windows?
- Where are performance problem areas? - Are there any specific issues in today's workload, why are these problems there, what is being done to fix these?
- What are the priorities, and do they change during a time window? - Which of the various workloads are most important and when?
- Are there priority conflicts? - And if so, who is going to make sure the right decisions are made on the real priorities?

Understanding the workload is a crucial phase! If your understanding is incorrect, your plans are incorrect and you will see issues popping up during the initial running of the workload. Poorly understood workloads might even drive you back to square zero and cause a lot of issues when a system (like an operational DW) is mission critical.

Creating and Implementing the Plan

Now that you know (in detail) the characteristics of your workload you can start to document it all and then implement this plan. It is recommended to document the details and reasoning for the decisions (as with all systems). Out of the documented plan you would create (already in Oracle speak, more later on the products):

Create the required Resource Plans:

For example: Nighttime vs. daytime, online vs. offline

Create the resource groups:

Map to users, application context or other characteristics

Map based on estimated execution time

Etc

Set the overall priorities:

Which resource group gets most resources (per plan/window) for IO, CPU, Parallel Processing

Cap max utilizations that these sessions can use on the system

Etc

Create thresholds:

Estimated execution times to determine in which group to run

Reject sessions if too much time, CPU or IO is required (both estimated and actual)

Downgrade (or upgrade) based on resources used

Set queuing thresholds for parallel statements

Etc

Create throttles:

Limit the number of active sessions

Limit degrees of parallelism

Limit the maximum CPU, IO that can be allocated

Etc

The above is just a small number of the things to consider when putting your plan into action and is mostly focused on Database Resource Manager and IO Resource Manager (IORM is Exadata only). Also consider working with Services and Server Pools and when you running several databases on a system consider instance caging.

Monitoring and Adjusting

Last but not least you will put the plan into action and now monitor your workloads. As the system runs you will adjust the implemented settings. Those adjustments come at various levels:

System Levels:

Memory allocations

Queuing Thresholds

Maximum Parallel Processes running

Server Pools

Etc

Resource Management Settings:

Increase or Decrease throttles and thresholds

Change the queuing guidelines

CPU and IO levels

Etc

All of these adjustments should be minor tweaks... if there are major changes required you should consider a second look at your original assumptions and workload definition.

Concepts for Concurrency

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. This chapter places some of the new Oracle Database 11g Release 2 features in that context.

The goal of Auto DOP

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.

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. So far there is nothing new here..., but do realize that anything serial (e.g. that stays under the threshold) goes straight into processing as is not considered in this paper.

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, let's 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 $4 * \text{Default DOP}$ (2 processes for a DOP, default value is $2 * \text{Default DOP}$, hence a default of $4 * \text{Default DOP}$).

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.

Understanding Parallel Statement Queuing

The previous section did discuss parallel statement queuing as it is a key player in a concurrent environment. The following is a much closer look at statement queuing and it should be understood that this is looking purely at Oracle Database 11.2.0.2 (patchset 1 on top of 11g Release 2).

A Step Back - When do you get this functionality?

Like I said, the statement queuing is really something introduced in 11.2.0.1, but I will refer to it with its enhancements in 11.2.0.2. To enable the new set of parallel features you must modify an `init.ora` parameter. Once you enable the new features you get a few more parameters that can be useful to work with if the default values prove they need some adjustments. These parameters form a hierarchy of sorts:

- | | |
|--|--|
| <ol style="list-style-type: none"> 1. Parallel_degree_policy = Manual <ol style="list-style-type: none"> a) None of the parameters have any impact | <div style="border: 1px solid black; background-color: yellow; padding: 5px;"> PX Features:
 • NONE </div> |
| <ol style="list-style-type: none"> 2. Parallel_degree_policy = Limited <ol style="list-style-type: none"> a) <code>Parallel_min_time_threshold = 10s</code> b) <code>Parallel_degree_limit = CPU</code> | <div style="border: 1px solid black; background-color: yellow; padding: 5px;"> PX Features:
 • Auto DOP </div> |
| <ol style="list-style-type: none"> 3. Parallel_degree_policy = Auto <ol style="list-style-type: none"> a) <code>Parallel_min_time_threshold = 10s</code> b) <code>Parallel_degree_limit = CPU</code> c) <code>Parallel_servers_target = Set to Default DOP on Exadata</code> | <div style="border: 1px solid black; background-color: yellow; padding: 5px;"> PX Features:
 • Auto DOP
 • Queuing
 • In-Memory </div> |

Figure 2. Parameter Hierarchy

Here we really only look at case #3 => Auto. When the database is set up for auto all new functionality is enabled. So you will get full Automatic DOP, full statement queuing and In-memory parallel execution.

Here we assume that the default, 10 second threshold set via `parallel_min_time_threshold` is fine.

Your Safety Net - Capping the maximum DOP

One way of using `parallel_degree_limit` is to cap the DOP for every statement (inserts, updates and selects) to a set number (as discussed in the previous section). However it is much more flexible to leverage Database Resource Manager for that. Consequently it would be better

to use `parallel_degree_limit` as your safety net. In other words, it is your safeguard to an extreme DOP on the system.

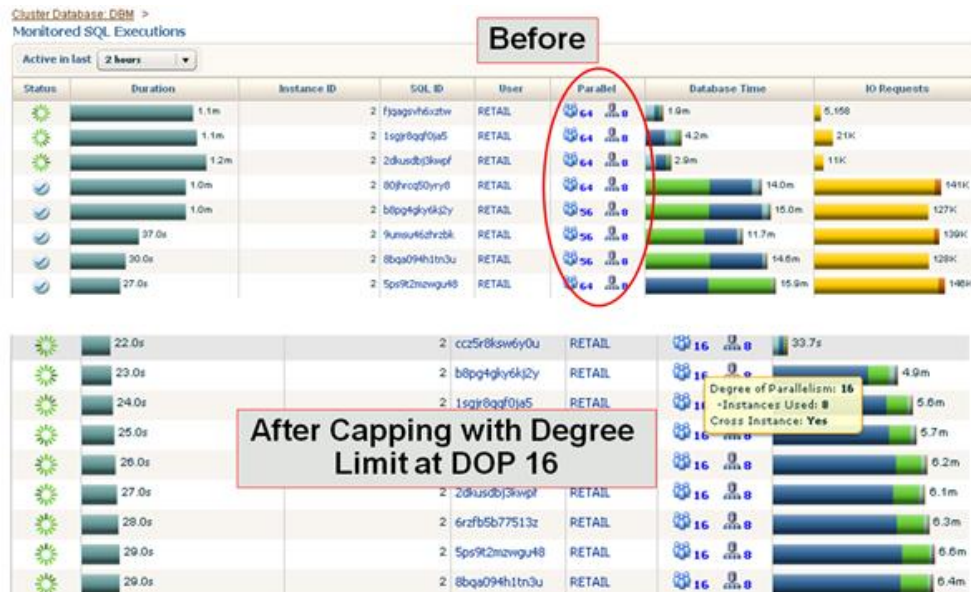


Figure 3. A quick look in SQL Monitor

The above is an extreme case where we have automatic DOPs generated showing 56 and 64 across a number of statements. Here we set `parallel_degree_limit` to 16, which is more akin to using it as a concurrency vehicle. It does however show the effect in Enterprise Manager and it does show there is no visual way of telling a parameter is kicking in!

For 11.2.0.2 it is best to think of `parallel_degree_limit` as the safety net and use the groups in Database Resource Manager to enable caps on the DOPs.

Know when to start queuing parallel statements

First of all, you should decide whether or not you want queuing and how aggressive you want to apply it in your system / your group.

That is because queuing comes with both upsides and downsides. The upside is that each statement will run with a computed or set DOP, no downgrades. That means that you run a statement faster than if you run it with a lower DOP (provided the DOP calculation is reasonably well done). It also means that you do not thrash the system by using too much resources.

The downside is an element of unpredictability. E.g. if the statement runs in 20 seconds today, it may run in 30 seconds tomorrow due to 10 seconds in the queue. An individual user may notice

these differences. Obviously, if you do not queue the users statement may simply get hammered and downgraded, making life much worse.

The theory is that the upside outweighs the downside and that - overall - a workload on your system runs better and at faster aggregate times.

Calculating minimal concurrency

To make the parameter settings a bit more tangible it pays to think of queuing as the point in time when you have enough statements running concurrently to accept wait times for new statements. Or differently put: find the optimal queuing point based on desired concurrency on your system. I call that minimal concurrency.

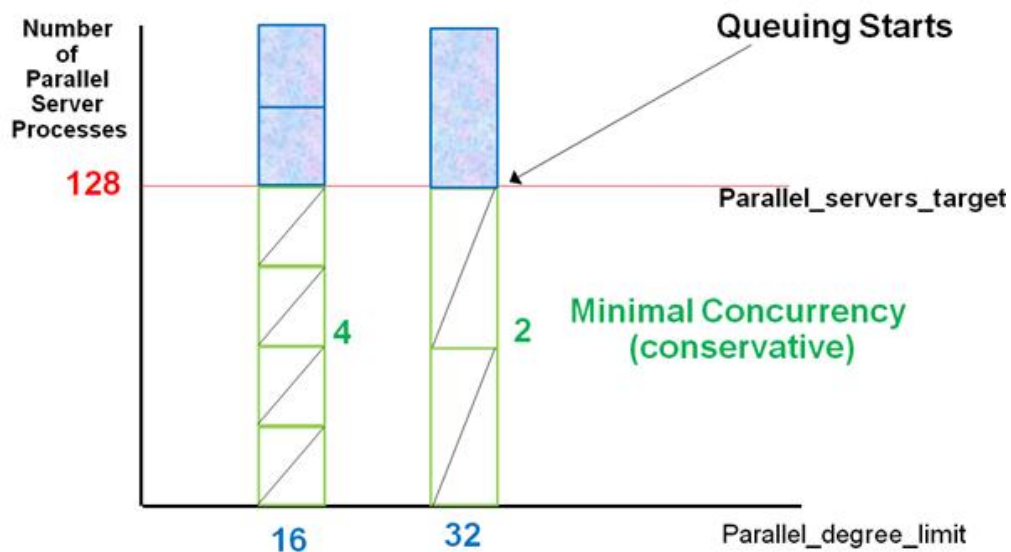


Figure 4. Understanding Concurrency

To visualize minimal concurrency consider two scenarios as shown in the graph above, one with a maximum DOP of 16, one with a maximum DOP of 32. Again, I used `parallel_degree_limit`, but you should do this by group and you will get the same calculation. By using the `parallel_degree_limit` it is just simpler to explain as there is only one value for the entire system.

In the case of a DOP of 16, with `parallel_servers_target` at 128, you can run at minimum 4 statements concurrently. When the cap is set to 32, you can run 2 statements concurrently.

There are two interesting things to note here.

One is that the above is a very conservative calculation in that we assume each statement runs in with that maximum allowed DOP. That is probably not the case, however if this is the case it may be worth elevating the cap a bit so statements can run faster with higher DOPs.

Secondly, the math does not quite add up...

The reason for my math being off is that the DOP is not the same as number of processes used to run this statement in parallel (or even better, it may not be the same - see [partition wise joins](#) for more on that). As Oracle uses producers and consumers the number of processes is often double the DOP. So for a DOP of 16, you often need 32 processes.

`Parallel_servers_target` is set in processes (not in DOP), so on the conservative side we should assume $2 * \text{DOP}$ as the number of processes (hence the diagonal bar in the little graph blocks). And then the math all of a sudden adds up again...

Since we are doing the math, this is the formula to use:

$$\text{minimal concurrency} = \frac{\text{Parallel_servers_target}}{\text{Parallel_degree_limit}} \times 0.5$$

Figure 5. The correct minimal concurrency

You will have to think about the minimal concurrency per group in Database Resource Manager. Next up therefore, a little bit on DBRM and how you can set up groups in plans and work with the concurrency, queuing and DOP caps per group.

Database Resource Manager (DBRM)

Before you get excited, this is only going to touch on a small aspect of DBRM, which helps with the parallel execution functionality mentioned above. But to place that in some context I will quickly put down the basics of DBRM in a picture so we all get on board on terminology and high level concepts.

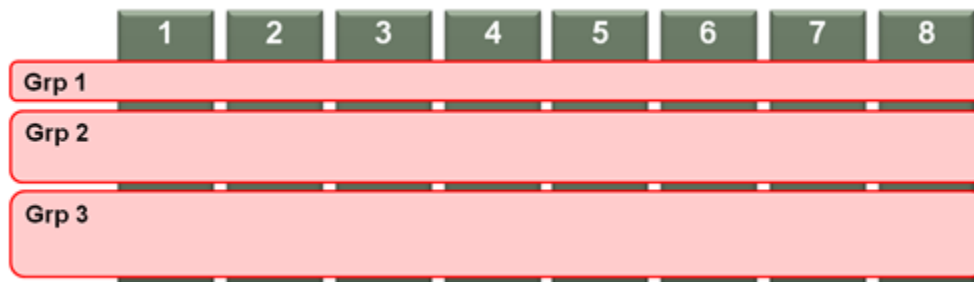


Figure 6. Logical division of a cluster

As a simple example, the above shows an 8-node RAC cluster divided into three resource groups. The cool thing about this is that as long as no one in Group 2 and 3 is working, group 1 can have the entire machine's resources. As soon as the others start to submit work, group 1 gets throttled back to where each other group get its rightful resources.

In DBRM the main entity is the resource plan. A resource plan consists of groups resource allocations and policies (known as directives and thresholds). Figure 7 shows a simple plan with three resource groups. The groups are called Static Reports, Tactical Queries and Ad-hoc Workload. Any request that comes into the database gets assigned to a specific group when this plan is active. Assignments can happen based on application context, based on Database user etc. Those are relatively static. The assignment can also happen based on estimated execution time, which is very dynamic.

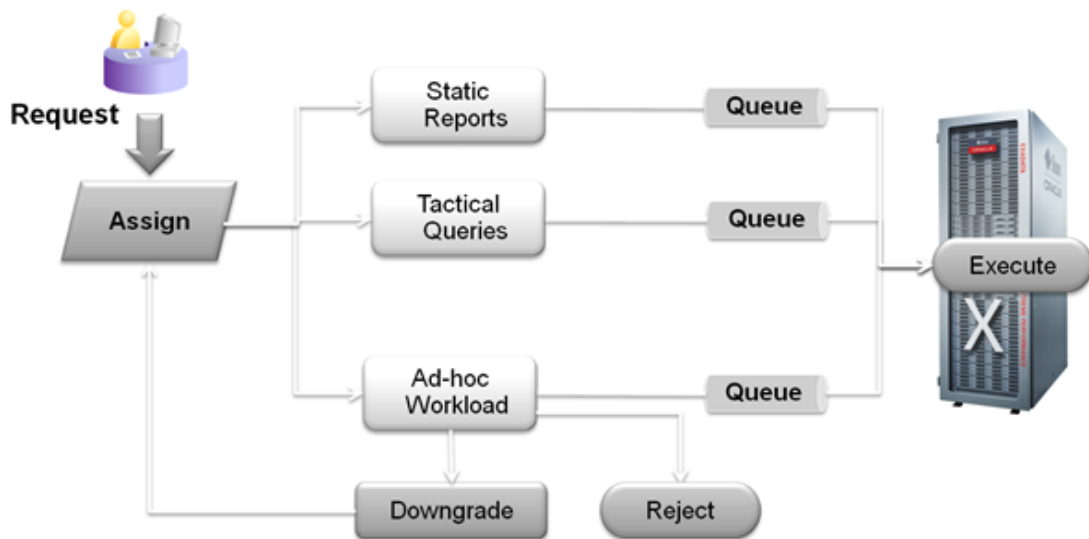


Figure 7. Basic DBRM workflow

In 11.2.0.2 each of the resource groups can have a separate parallel statement queue (as shown above).

Each group can have properties set to downgrade a statement to another group, or to reject the query from running in the first place if its estimated execution time is too long. You can also determine that a query gets terminated or cancelled if runs too long, this of course being evaluated on actual elapsed time. You'd want to catch the error in your application to make sure it is handled well:

ERROR at line 1:

ORA-12801: error signaled in parallel query server P000

ORA-00040: active time limit exceeded - call aborted

The Enterprise Manager screen below shows some of these settings.

The screenshot shows the Oracle Enterprise Manager interface for a Resource Plan named PEAK_PLAN. The plan is described as being for online time that heavily prioritizes short-running queries. It is active on instances DBM1, DBM2, DBM3, DBM4, DBM5, DBM6, DBM7, and DBM8. Automatic Plan Switching is disabled, and it is not a subplan.

Resource Allocations

Group/Subplan	Current CPU Utilization	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8
LONG_SQL_GROUP				50					
MEDIUM_SQL_GROUP		30							
OTHER_GROUPS				100					
SHORT_SQL_GROUP		70							
SYS_GROUP			50						

Directive Values

Group	Maximum Degree of Parallelism	Maximum Number of Active Sessions	Activation Timeout (sec)	Maximum Undo Space (KB)	Maximum Estimated Execution Time (sec)	Max Idle Time (sec)	Max Idle Time if Blocking Another Session (sec)	Maximum Percentage of Parallel Sessions Target
LONG_SQL_GROUP	64	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	50
MEDIUM_SQL_GROUP	64	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	50
OTHER_GROUPS	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED
SHORT_SQL_GROUP	128	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED
SYS_GROUP	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED

Threshold

Group	Execution Time Limit (Sec)	I/O Limit (MB)	I/O Request Limit (Requests)	Action	Revert after call?	Use estimate?
LONG_SQL_GROUP	1200	UNLIMITED	UNLIMITED	Kill This Session	<input type="checkbox"/>	<input type="checkbox"/>
MEDIUM_SQL_GROUP	300	UNLIMITED	UNLIMITED	Switch to Group LONG_SQL_GROUP	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
OTHER_GROUPS	UNLIMITED	UNLIMITED	UNLIMITED		<input type="checkbox"/>	<input type="checkbox"/>
SHORT_SQL_GROUP	180	UNLIMITED	UNLIMITED	Switch to Group MEDIUM_SQL_GROUP	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SYS_GROUP	UNLIMITED	UNLIMITED	UNLIMITED		<input type="checkbox"/>	<input type="checkbox"/>

Figure 8. DBRM Overview Screen

Here we will focus on the directives for parallel execution.

DBRM and Parallel Execution Management

Parallel Execution directives focus on both limiting the maximum DOP and on managing the statement queue as is shown here:

Cluster Database: DBM > Resource Plans >
 Edit Resource Plan: PEAK_PLAN

Actions: Create Like

General **Parallelism** Session Pool Undo Pool Thresholds Idle Time

Specify a limit on the degree of parallelism for any operation issued by this consumer group, a limit on the total number of parallel servers that can be used by all sessions in maximum time a parallel statement can be queued.

Group	Max Degree of Parallelism	Max Percentage of Parallel Servers Target	Parallel Queue Timeout
LONG_SQL_GROUP	64	25	UNLIMITED
MEDIUM_SQL_GROUP	64	50	UNLIMITED
OTHER_GROUPS	16	25	UNLIMITED
SHORT_SQL_GROUP	128	UNLIMITED	UNLIMITED
SYS_GROUP	UNLIMITED	UNLIMITED	UNLIMITED

General **Parallelism** Session Pool Undo Pool Thresholds Idle Time

Figure 9. Managing Statement Queuing and DOP

Max Degree of Parallelism

This is what we should be using to cap the DOP on a system. Rather than using `parallel_degree_limit` which is system wide, here we specify the cap on a per resource group basis. Dividing your workload into groups allows you to specifically dictate what max DOP a certain group can run with, without interfering with other queries. Just to clarify, the numbers assigned do **not** need to add up to any given total (like `max_parallel_servers`).

Parallel Queue Timeout

Slightly out of order, but it is much simpler than the next topic... This setting drives when a statement is ejected from the queue because it has been waiting too long. UNLIMITED means it will wait an unlimited number of seconds (also known as eternity). The statement will "error out" after reaching the time limit.

Max Percentage of Parallel Servers Target

This is a per group threshold to queuing. As the name implies, on a per group basis you say how much of the space before we start queuing is given to this group. If we have a setting of `parallel_servers_target = 128` the above allocates 25% of that 128 to the LONG_SQL_GROUP. Translated, this means that once 32 processes are in use by sessions running within LONG_SQL_GROUP next sessions within this group will be queued.

The total does not have to add up to 100%. In fact, UNLIMITED above equals to setting it to 100% allowing this group to only queue processes hit 128 (using the above numbers).

This over allocation has some interesting effects. The first is that it gives each group a certain leeway allowing it to queue a little later when the overall system is not busy. E.g. when nothing within MEDIUM and SHORT runs above, queuing will start no earlier than with 128 (e.g. 100%) processes in use for SHORT. When more processes run in other groups these groups use up their quota and SHORT will start queuing a bit sooner. That is because of the second

phenomenon of never going over the limit set in `parallel_servers_target` for the system.

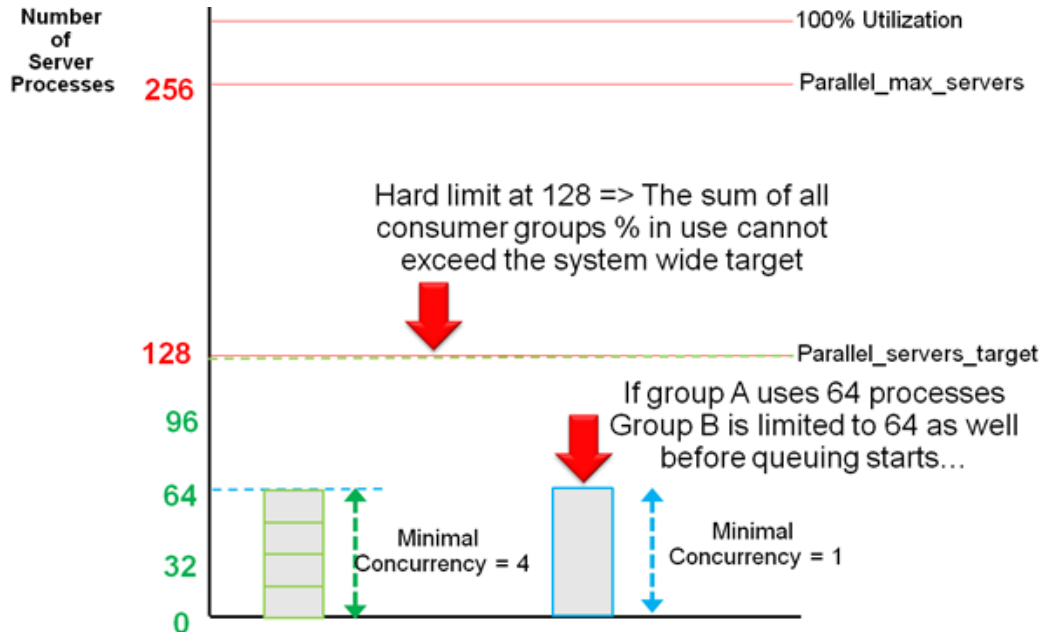


Figure 10. Overallocation does not overrule system parameters

The graph above (not always to scale!) shows this over allocation and capping in an extreme form. The blue group (group A) is allowed to use max percentage parallel servers target = unlimited, yet because group B (the green guys) are already using their maximum quota group A will start queuing sooner than "expected".

When using more groups it therefore pays to consider raising the bar and increasing the overall head room for queuing.

Leveraging Services

A great many systems today leverage services to divide a set of resources. Services works nicely with both DBRM and with statement queuing giving you another level of working with resource management.

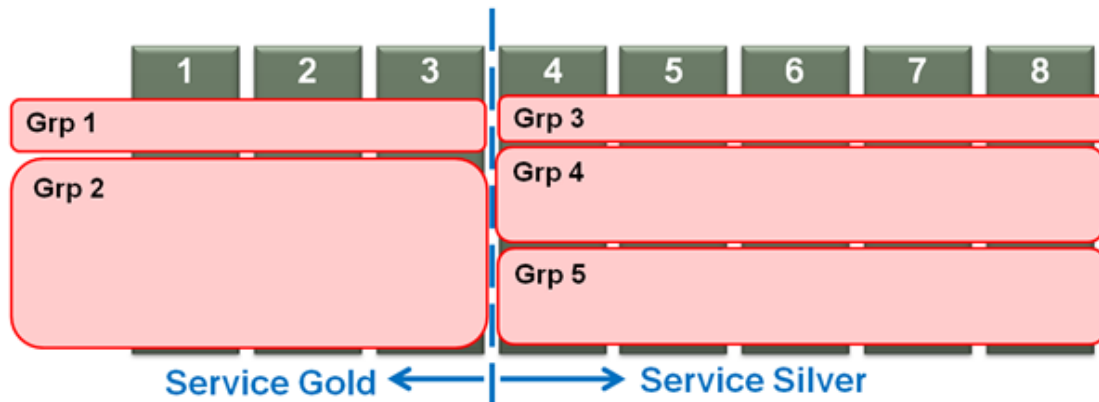


Figure 11. Working within Services

Consider two services created that divide the 8-node RAC system into 2 services called Gold and Silver. You can now layer the groups on top of the services. This allows you to separate the workloads in a physical environment (e.g. log onto service gold or silver) and then manage various consumer groups within that service.

By using server pools you can choose to change the service "size" and add or revoke a node from a service at specific times (for example to run a large ETL load on service Gold you may want to add a node to it).

Statement queuing gets still set within the group. But you must keep in mind that the service has only a limited number of nodes to its disposal and therefore a reduced number of processes in `parallel_servers_target` for the service. Each node has its number for `parallel_servers_target` and the aggregate for the whole system is `Node count * parameter value`. For the service it is `node count within service * parameter value`.

Setting up your system

As you can see from the above, there are a lot of ways to create a workload management system. The crucial piece is to understand your workload and then set the system up to handle such a workload. Different workloads require different solutions!

For example the set up for a system that divides a data warehouse workload into short running, medium running and long running queries is different from a system that has OLTP style and DW style workloads.

The first example would use dynamic assignment to consumer groups of queries based on estimated execution time. The second would probably leverage a user/application driven mapping to consumer groups.

The latter may use a very wide range of parallel settings. OLTP would have very low DOP caps, almost never queue and get the high priority when on-line orders are being placed - e.g. maximize

concurrency. The former would potentially focus on overall throughput and allow queuing to make sure few statements are suffering from too much capping.

Working Examples

To better understand the theory, the following are two examples of a real running workload solution. All are geared towards the mixed workload scenario we are discussing in this paper.

Example 1: Multi-plan DBRM Workload Management

In DBRM I have created 2 plans, one called batch_plan and one called daytime_plan. The batch_plan favors long running queries from the user RETAIL and gives the resource group called BATCH_PLAN (that is linked to RETAIL - again for simplicity I use a 1:1 mapping to a user) 70% of the CPU capacity and gives a resource group called RT_CRITICAL 30% of the CPU capacity. RT_CRITICAL is linked to a single user called... RT_CRITICAL which will run our highly critical queries for this environment.

The idea is that we are running a heavy workload with long running queries (batch type of stuff) within user RETAIL.

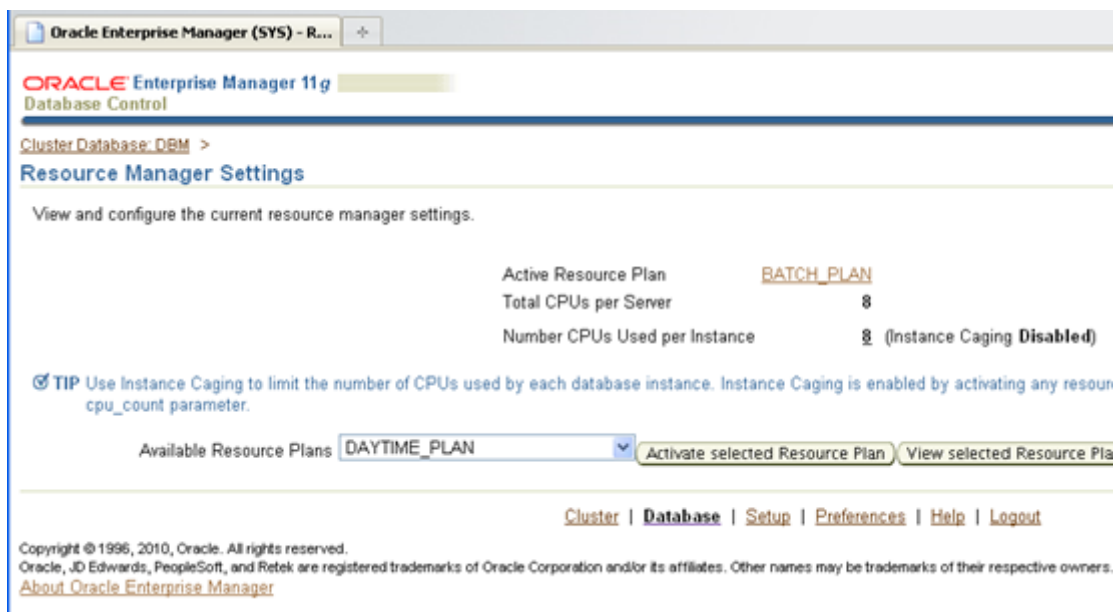


Figure 12. Two plans managing this workload

During the time window where we are running these batch like queries, we have the BATCH_PLAN enabled as is shown above. This means that with the current settings our critical queries run by RT_CRITICAL are squeezed to only 30% of CPU capacity. We have other throttles in place, but more about that later. First we will have a look at some of the query profiles and the result on our system of individual queries.

Running a Single Large Query

To show some of the characteristics for an individual query we run it with the BATCH_PLAN enabled and with nothing else running on the system.



Figure 13. SQL Monitor showing a query

We can see a couple of things, first off all the query is capped at DOP 64, running across 8 nodes of the database machine. Currently the query is in progress and clicking on the query will show the execution plan in action. This allows you to monitor long running queries in EM. The tool we are using here is called SQL Monitor.

SQL Monitor shows us the progress and the plan, of which we see a part here:

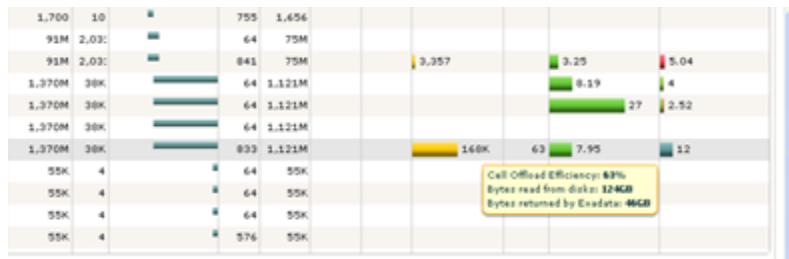


Figure 14. Cell Offload Efficiency on Exadata

As you can see this plan is a real query doing real work. You can also see that for some of the data / IO pieces we collect information on the Exadata efficiency. Here we achieve 63% offload efficiency. That means rather than moving 126GB to the compute nodes we only move 46GB.

We actually read that data with the peak capacity of my V1 hardware (note a V2 machine will push about 21GB or so per second), which is 14GB/sec as shown here:

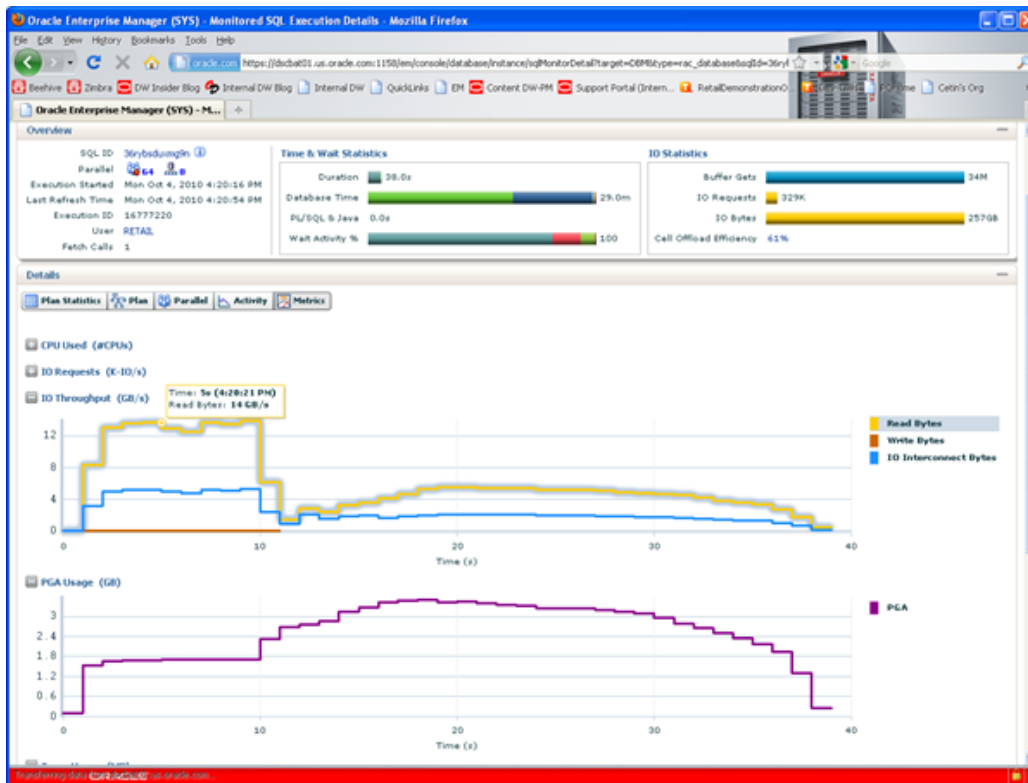


Figure 15. IO and Memory footprint of our query

You can see that we read at 14GB/sec in the top graph - yellow line. What is also interesting is that we see the memory build up underneath. E.g. as we are reading at peak levels we are building any constructs in memory and slowly increase the PGA usage. Once the IO is mostly done PGA starts going up and we work in PGA.

Critical Queries

Now that we have an idea about our batch queries, let's look at some characteristics of the critical queries. We have 5 of them in this scenario, and if we run them all by themselves they complete in around 5 seconds or so on average. Note nothing else is running on the system, so even while we have BATCH_PLAN active these 5 queries can use all CPU resources. This is because DBRM allows a group to use more resource as long as the other groups are not working on the system.

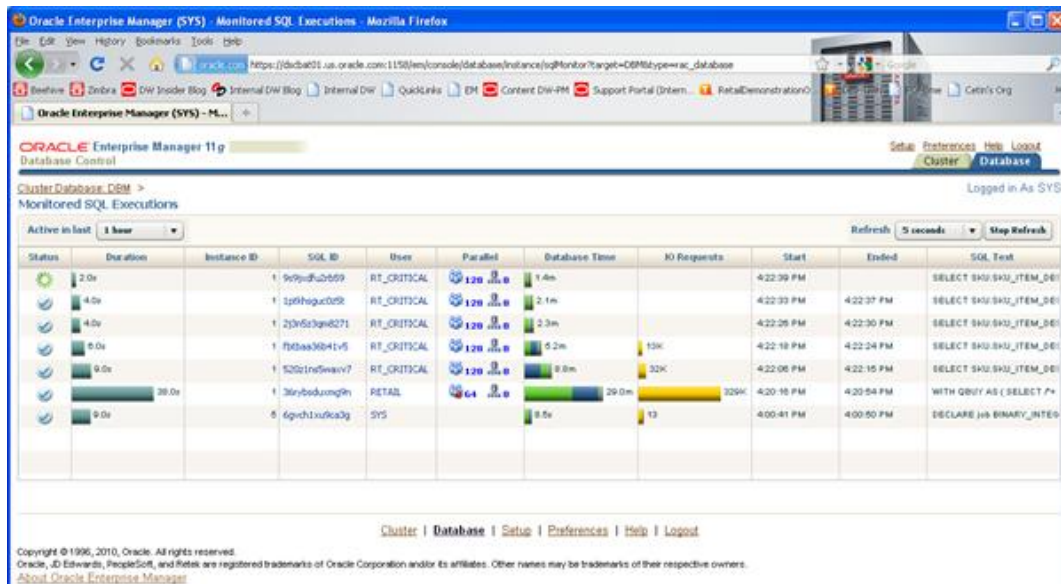


Figure 16. Short-running queries

The Goal of the Case Study

Now, the entire idea is to load up the machine with batch queries and show the effect of this load on the critical queries. With the batch plan, the critical queries will be squeezed for resources and show a large variation in runtime. We will then switch to the daytime_plan, and will see that the critical queries run as if nothing else is going on on the system.

To understand what is going on at the system and plan level let's have a look at the actual plans we will be using here.

BATCH_PLAN

The BATCH_PLAN is the plan that favors the analytical or large batch queries. Here we can see (in order from top to bottom) the settings (in edit mode) for CPU, Parallelism and Thresholds:

ORACLE Enterprise Manager 11g
Database Control

Cluster Database: DEM > Resource Plans >
Edit Resource Plan: BATCH_PLAN

Actions: Create Like Go Show SQL Revert Apply

General Parallelism Session Pool Undo Pool Thresholds Idle Time

A Resource Plan contains directives that specify how resources are allocated to Consumer Groups. For each Consumer Group, a directive specifies the amount of CPU resources are allocated. It also specifies limits, such as the maximum degree of parallelism, execution time, and amount of I/O, that each session in the Consumer Group can consume. You can enable a Resource Plan manually or automatically, using Scheduler Windows.

Plan: BATCH_PLAN
Description: Batch Bias - Plan for off line hours in Retail Environment

Instances: DBM1, DBM2, DBM3, DBM4, DBM5, DBM6, DBM7, DBM8 (Edit)
If any instances are in this list, the Plan is active for those instances.

Resource Allocations
Mode: Percentage Advanced

Group/Subplan	Max Utilization Limit	Percentage
OTHER_GROUPS		
RT_ANALYTICS		70
RT_CRITICAL		30

General Parallelism Session Pool Undo Pool Thresholds Idle Time

Actions: Create Like Go Show SQL Revert Apply

Cluster | Database | Setup | Preferences | Help | Logout

Copyright © 1996, 2010, Oracle. All rights reserved.
Oracle, JD Edwards, PeopleSoft, and Retek are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.
About Oracle Enterprise Manager

Figure 17. Setting CPU Percentages

ORACLE Enterprise Manager 11g
Database Control

Cluster Database: DEM > Resource Plans >
Edit Resource Plan: BATCH_PLAN

Actions: Create Like Go Show SQL Revert Apply

General Parallelism Session Pool Undo Pool Thresholds Idle Time

Specify a limit on the degree of parallelism for any operation issued by this consumer group, a limit on the total number of parallel servers that can be used by all sessions in this consumer group, and the maximum time a parallel statement can be queued.

Group	Max Degree of Parallelism	Max Percentage of Parallel Servers Target	Parallel Queue Timeout
OTHER_GROUPS	UNLIMITED	UNLIMITED	UNLIMITED
RT_ANALYTICS	64	70	UNLIMITED
RT_CRITICAL	UNLIMITED	UNLIMITED	UNLIMITED

General Parallelism Session Pool Undo Pool Thresholds Idle Time

Actions: Create Like Go Show SQL Revert Apply

Cluster | Database | Setup | Preferences | Help | Logout

Copyright © 1996, 2010, Oracle. All rights reserved.
Oracle, JD Edwards, PeopleSoft, and Retek are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.
About Oracle Enterprise Manager

Figure 18. Setting Parallel Directives

ORACLE Enterprise Manager 11g
Database Control

Cluster Database: DBM > Resource Plans >
Edit Resource Plan: BATCH_PLAN

Actions: Create Like [v] Go Show SQL Revert Apply

General Parallelism Session Pool Undo Pool **Thresholds** Idle Time

Specify the time duration or the resource limits under which a session can execute in a consumer group. If any of the limits are exceeded, the session can be switched to another consumer group, the session's SQL operation can be canceled, or the session can be killed.

Group	Execution Time Limit (Sec)	I/O Limit (MB)	I/O Request Limit (Requests)	Action	Revert after call?	Use estimate?
OTHER_GROUPS	UNLIMITED	UNLIMITED	UNLIMITED		<input type="checkbox"/>	<input type="checkbox"/>
RT_ANALYTICS	UNLIMITED	UNLIMITED	UNLIMITED		<input type="checkbox"/>	<input type="checkbox"/>
RT_CRITICAL	UNLIMITED	UNLIMITED	UNLIMITED		<input type="checkbox"/>	<input type="checkbox"/>

General Parallelism Session Pool Undo Pool **Thresholds** Idle Time

Actions: Create Like [v] Go Show SQL Revert Apply

Cluster | Database | Setup | Preferences | Help | Logout

Copyright © 1996, 2010, Oracle. All rights reserved.
Oracle, JD Edwards, PeopleSoft, and Retek are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.
[About Oracle Enterprise Manager](#)

Figure 19. Setting Thresholds and Actions

As you can see you can modify and create these plans within Enterprise Manager, under the Server Tab in DB Control. We have set CPU limits at 70% for RT_ANALYTICS. We have set a maximum DOP for RT_ANALYTICS at 64, and give it 70% of the available parallel processes set in `parallel_servers_target`. This means it starts queuing at 70% of $8 * 128$.

In this plan we have not worked with any thresholds. Again this is for simplicity reasons. If you look at the Thresholds it is important to understand that the Use Estimate? column signifies that we use the optimizer estimate for execution time to determine what to do. So before the query executes you can decide on thresholds and for example move something from RT_CRITICAL into RT_ANALYTICS.

DAYTIME_PLAN

The following is the overview of the DAYTIME_PLAN. If it needs any editing you press Edit and get the tabbed version of the plan as we saw earlier in our BATCH_PLAN screenshots.

ORACLE Enterprise Manager 11g
Database Control
Cluster Database: DEM > Resource Plans >
View Resource Plan: DAYTIME_PLAN
Logged in As SYS

Plan: **DAYTIME_PLAN**
Description: **Short Query Bias - Online plan for Retail Environment**
Instances: **(none)**
Automatic Plan Switching Enabled: **false**
Is Subplan: **false**
Associated Scheduler Window(s):

Resource Allocations

Group/Subplan	Max Utilization Limit	Percentage
OTHER_GROUPS		
RT_ANALYTICS		30
RT_CRITICAL		70

Directive Values

Group	Max Degree of Parallelism	Max Percentage of Parallel Servers Target	Parallel Query Timeout	Max Number of Active Sessions	Activation Queue Timeout (sec)	Max Undo Space (KB)	Max Estimated Execution Time (sec)	Max Idle Time (sec)	Max Idle Time if Blocking Another Session (sec)
OTHER_GROUPS	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED
RT_ANALYTICS	16	30	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED
RT_CRITICAL	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED

Thresholds

Group	Execution Time Limit (Sec)	I/O Limit (MB)	I/O Request Limit (Requests)	Action	Revert after call?	Use estimate?
OTHER_GROUPS	UNLIMITED	UNLIMITED	UNLIMITED	Cancel SQL	<input type="checkbox"/>	<input type="checkbox"/>
RT_ANALYTICS	UNLIMITED	UNLIMITED	UNLIMITED		<input type="checkbox"/>	<input type="checkbox"/>
RT_CRITICAL	30	UNLIMITED	UNLIMITED	Switch to Group RT_ANALYTICS	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Cluster | Database | Setup | Preferences | Help | Logout

Figure 20. Overview of the DayTime plan

As we can see in the DAYTIME_PLAN we did set thresholds but these are based on actual execution times (we left the Use Estimate? box unchecked!).

Within the DAYTIME_PLAN we reversed the CPU settings and now are giving only 30% of CPU resources to the RT_ANALYTICS group. RT_ANALYTICS is also capped at a much lower DOP (16 instead of 64) and will start queuing at 30% of the total number in `parallel_servers_target`.

Running the Batch

With the BATCH_PLAN on, we will first start a load on the system. The system is always loaded with 10 to 12 high octane queries like the one we looked at when it ran by itself.

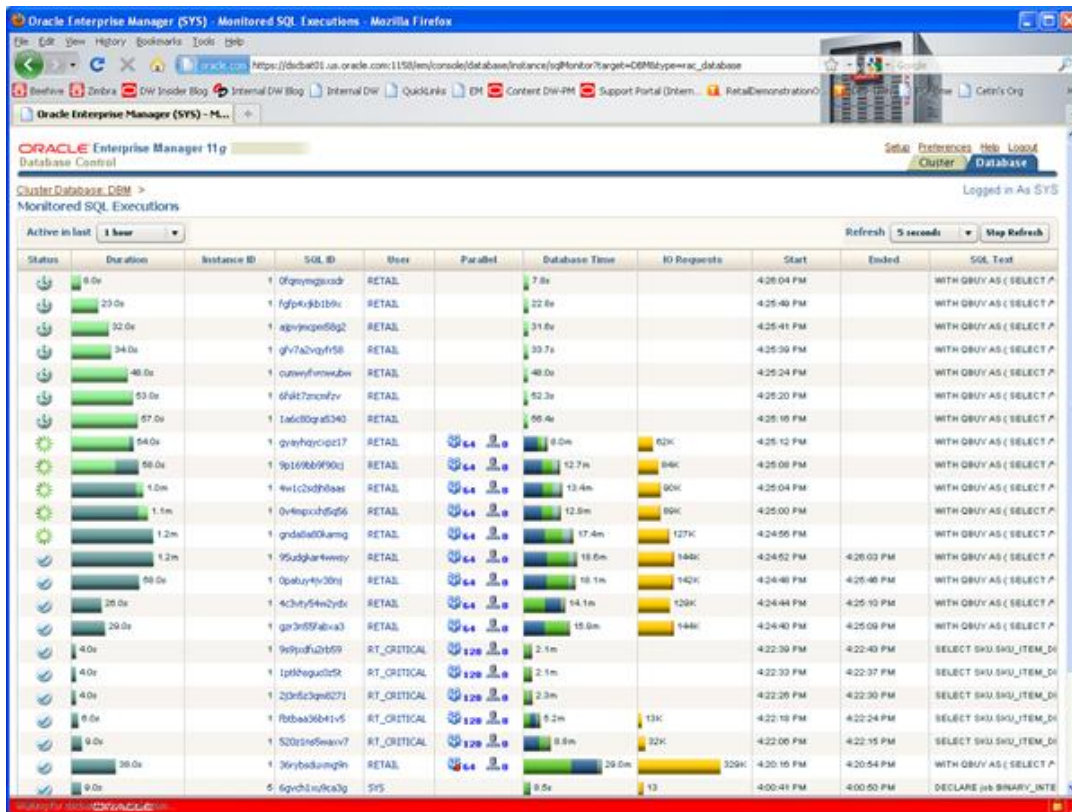


Figure 21. A running system under the batch plan

SQL Monitor above shows what is going on. We have five queries running at DOP 64 (with 128 processes each). We see that subsequent statements are being queued to ensure we do not completely overload the system and use up all parallel processes.

This statement queuing is an important workload management tool as it stops statements from being downgraded to serial (which happens once we run out of parallel processes).

With the system loaded up with queries, we are starting our critical queries. As expected, the critical queries struggle to get enough resources. They are not queued in this case and get the appropriate DOP, but are starved for CPU (and IO).

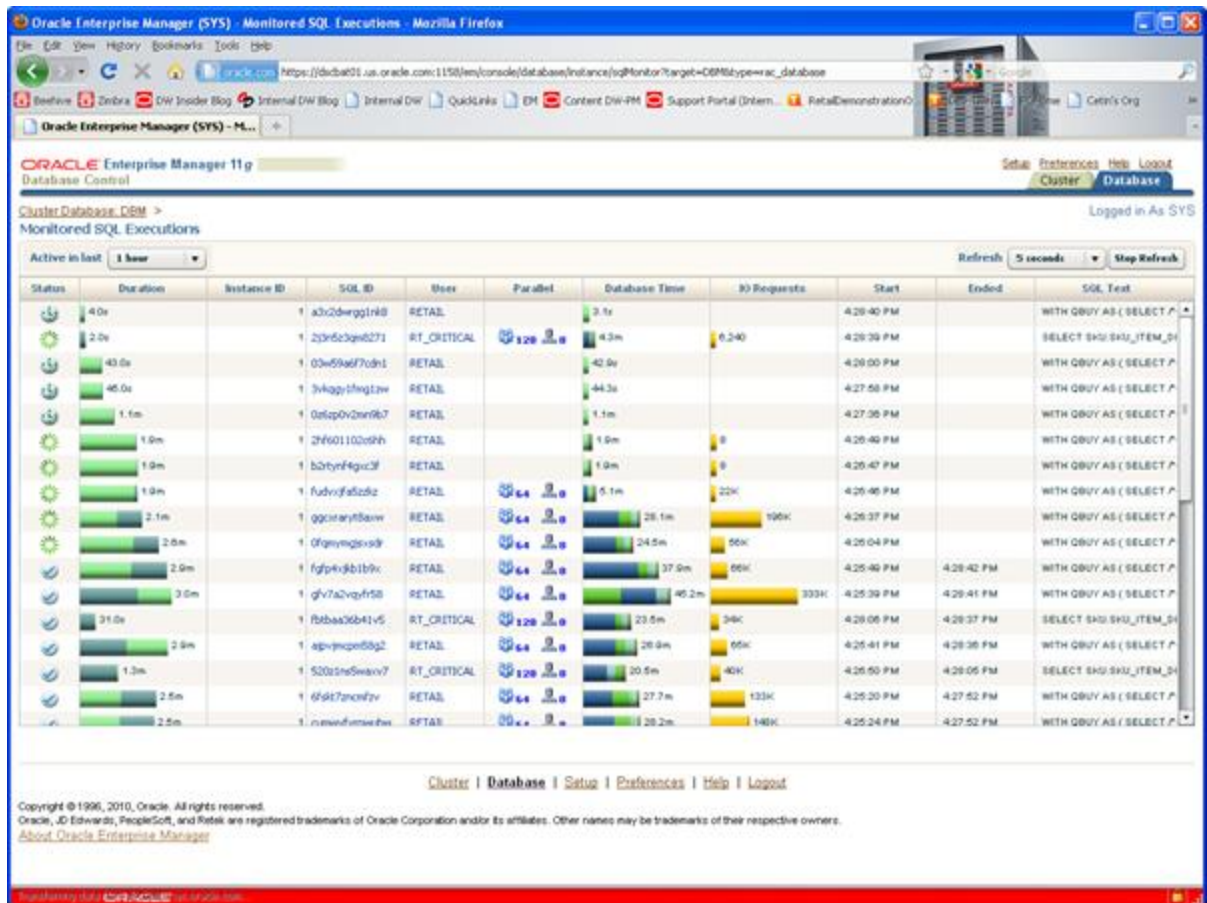


Figure 22. Short running queries struggle for resources - as planned

If you look closely above, you see that an RT_CRITICAL query completed in 1.3 minutes, one completed in 31 seconds and a third one is running. Eventually the latter queries would complete in around 25 seconds or so. This is of course a far cry from the 5 seconds on average. But it is expected as we place full emphasis on the batch jobs.

Next we switch the plan to the DAYTIME_PLAN. In our case we do this via the UI, but in reality this would be scheduled, which is of course supported via the database scheduler.

A Better Run

After we switch the plan we can see a couple of things. First of all we see the new DOP cap for any statement within RT_ANALYTICS. The cap is now set to 16:

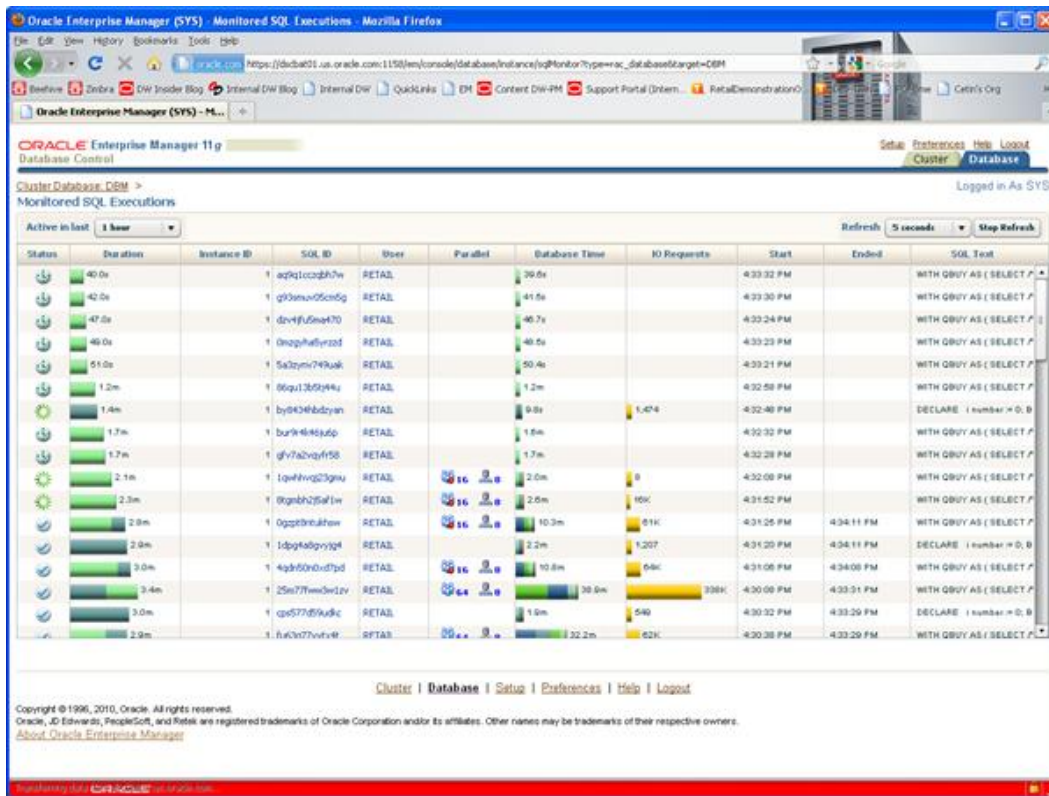


Figure 23. Different DOPs and better runs

As you can see we still have statement that completed under the old plan with DOP 64, but the new statements that started after we change the plan are now showing a cap of 16.

If we now run the critical statements we will see that they run very near their original runtimes of 5 seconds:

Status	Duration	Instance ID	SQL ID	User	Parallel
	20.0s	1	6pmt72f02z8r	RETAIL	
	54.0s	1	b145trxvhduvw	RETAIL	
	1.8m	1	2w49rww0k04n0	RETAIL	
	1.8m	1	6jnwzywxqq22z	RETAIL	
	2.5m	1	aq9q1cczqbh7w	RETAIL	
	2.5m	1	g93smuv05cm5g	RETAIL	
	2.8m	1	dzv4jfu5ma470	RETAIL	
	2.8m	1	0mzgyha5yrzdz	RETAIL	
	2.8m	1	5a3zynv749uak	RETAIL	
	3.0m	1	86qu13b5tj44u	RETAIL	
	3.2m	1	by8434hbdzyan	RETAIL	
	3.5m	1	bur9r4k46ju6p	RETAIL	16 8
	3.5m	1	gfv7a2vqyfr58	RETAIL	16 8
	5.0s	1	9s9jxdfu2rb59	RT_CRITICAL	128 8
	4.0s	1	1ptkhsquc0z5t	RT_CRITICAL	128 8
	4.0s	1	2j3n5z3qm8271	RT_CRITICAL	128 8

Figure 24. Running critical statements at optimal speed

Note that I cropped the statement to show the queuing, the caps and the critical runtimes.

Conclusion

While this scenario is relatively simple, you can quickly see the power of the functionality. Here we protect statements from downgrades to serial through queuing. We ensure the important workloads get the appropriate resource allocation when it matters (here we did it by time windows), and we can see how a heavy load can be managed to ensure critical work gets done.

By using the parallelism in 11.2.0.2, Database Resource Manager and SQL Monitor (which is part of the EM Tuning pack!) you can manage your mixed workloads the way you want to manage them.

Dynamic Management based on Execution Time

How do I manage a workload if I do not have any users to map to consumer groups as shown before? Well you could use application context and other nice things in DBRM, but really, what if you want this to be driven by the amount of work to be done?

That is what this example shows, a way of dynamically moving queries from one resource plan to the other BEFORE it executes. Doing the switch before the statement actually starts is beneficial in many cases. It certainly helps with getting the right resources allocated from the start. If all goes well, it also helps with runaway query prevention by choosing the right group (resource profile) for the statement.

The following code (for credits read on) creates a dynamic switching system based on the optimizer's estimate of the execution time. We create 3 resource groups (one of which is OTHER_GROUPS - default on each system) which are assigned a query time. One group is for "long running" queries, the other for "medium" and one more for "short". All these are in "" because it is up to you (and me) to define what short means for you. So if your groups are 10, 100 and 1000 seconds then that is what we use as switch boundaries.

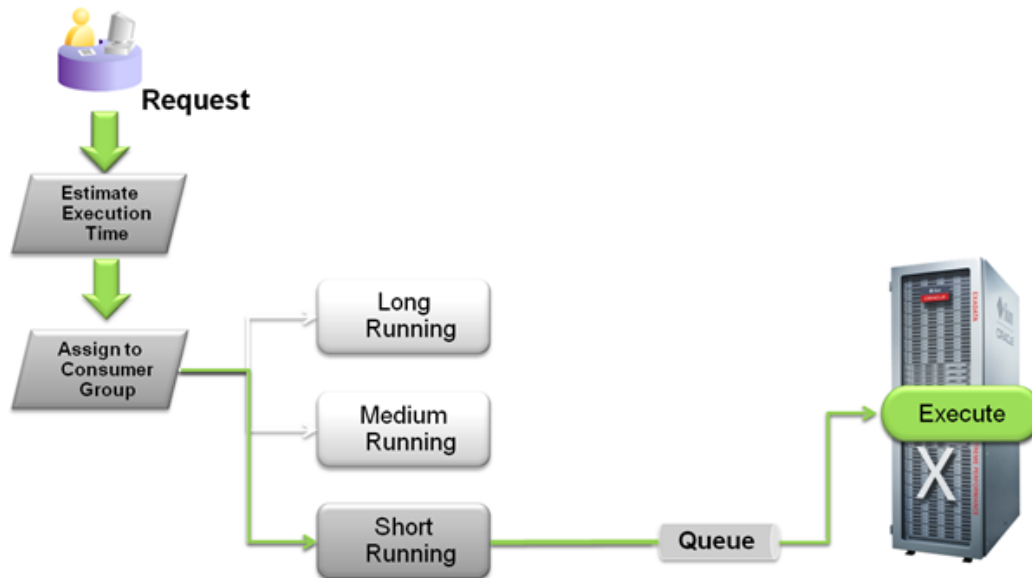


Figure 25. Use DBRM to dynamically switch based on execution times

When a user submits a statement the database optimizer determines the estimated duration of that statement. Based on that estimate DBRM then assigns a query to a specific resource group, the statement then runs with the appropriate resources (parallel statement queuing, max parallel degree, CPU and IO etc.).

To build something like this, have a look at this code example:

```

begin
dbms_resource_manager.create_pending_area();
  /* Create consumer groups.
   * By default, users will start in OTHER_GROUPS, which is
   automatically
   * created for every database.
   */

```



```

dbms_resource_manager.create_consumer_group(
    'MEDIUM_SQL_GROUP',
    'Medium-running SQL statements, between 1 and 15 minutes.
Medium priority.');
```

```

dbms_resource_manager.create_consumer_group(
    'LONG_SQL_GROUP',
    'Long-running SQL statements of over 15 minutes. Low
priority.');
```

```

/* Create a plan to manage these consumer groups */

dbms_resource_manager.create_plan(
    'REPORTS_PLAN',
    'Plan for daytime that prioritizes short-running queries');
```

```

dbms_resource_manager.create_plan_directive(
    'REPORTS_PLAN', 'SYS_GROUP', 'Directive for sys activity',
    mgmt_p1 => 100);

dbms_resource_manager.create_plan_directive(
    'REPORTS_PLAN', 'OTHER_GROUPS', 'Directive for short-running
queries',
    mgmt_p2 => 70,
    parallel_degree_limit_p1 => 4,
    switch_time => 60, switch_estimate => TRUE, switch_for_call
=> TRUE,
    switch_group => 'MEDIUM_SQL_GROUP');
```

```

dbms_resource_manager.create_plan_directive(
    'REPORTS_PLAN', 'MEDIUM_SQL_GROUP', 'Directive for medium-
running queries',
    mgmt_p2 => 20,
    parallel_target_percentage => 80,
    switch_time => 900, switch_estimate => TRUE, switch_for_call
=> TRUE,
    switch_group => 'LONG_SQL_GROUP');
```

```

dbms_resource_manager.create_plan_directive(
    'REPORTS_PLAN', 'LONG_SQL_GROUP', 'Directive for medium-
running queries',
    mgmt_p2 => 10,
    parallel_target_percentage => 50,
    parallel_queue_timeout => 14400);

dbms_resource_manager.submit_pending_area();
end;
/

/* Allow all users to run in these consumer groups */

exec dbms_resource_manager_privs.grant_switch_consumer_group( -
    'public', 'MEDIUM_SQL_GROUP', FALSE);
```

```
exec dbms_resource_manager_privs.grant_switch_consumer_group( -  
  'public', 'LONG_SQL_GROUP', FALSE);
```

The time based approach allows a system to be set up based on the work a statement will do and because it does not rely on a user to group mapping it is very flexible and handles a workload with a large number of concurrent sessions with limited setup requirements.

Summary

As we have seen in the above examples Oracle Database 11g Release 2 offers a number of ways to control and manage a complex, mixed workload for data warehousing. This capability, added to the rich data warehouse functionality in general, allows an Oracle data warehouse to handle highly demanding and complex workloads.



Parallel Execution and workload management
for an operational data warehouse
October 2010

Author: Jean-Pierre Dijcks

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com



Oracle is committed to developing practices and products that help protect the environment

Copyright © 2010, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.