Workload Management for an Operational Data Warehouse

Jean-Pierre Dijcks
Oracle
Redwood City, CA, USA

Keywords:

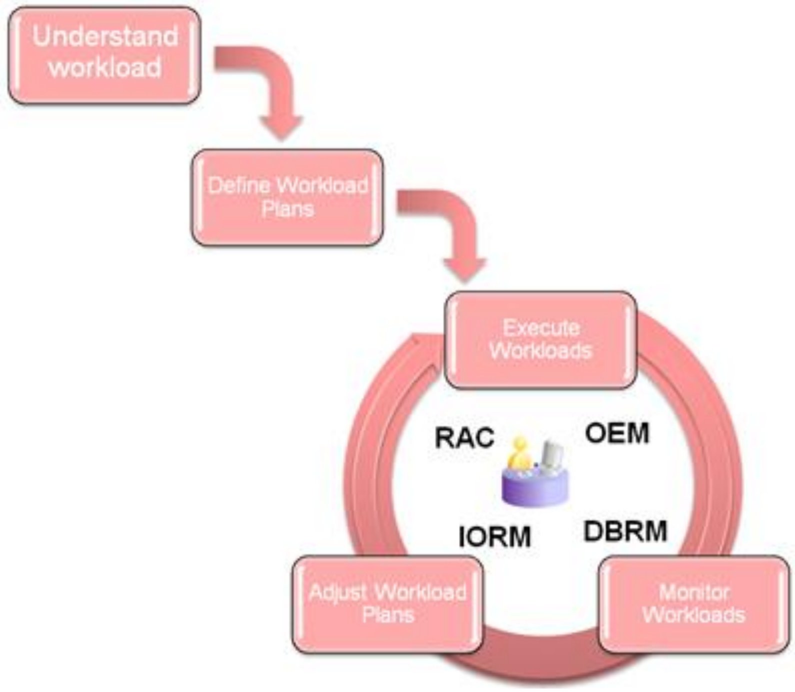
Workload management, data warehouse, database, parallel execution

Introduction

A lot can be said about setting up workload management, so expect this to be one of the many pieces to read on this topic. Let's start from the beginning, with understanding workloads and a framework of setting up such a management infrastructure.

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.



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 going back to drawing board and understand what the issues with your plan are.

Dealing with Concurrency and Parallel Processing

For basic understanding make sure you have read the initial post. The focus there is on Auto DOP and queuing, which is to some extend the focus here. But now I want to discuss the concurrency a little and explain some of the relevant parameters and their impact, specifically in a situation with concurrency on the system.

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

- Don't throttle down a DOP because other statements are running on the system
- 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 the rest of this post.

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 ain't 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 Maximum DOP for a Statement
Now that you know how to control how many of your statements are considered to run in parallel, lets talk about the specific degree of any given statement that will be evaluated. As the initial post 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 post...
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.

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