An Oracle White Paper
August 2010

Oracle Grid Engine: An Overview

ORACLE

Executive Overview

Oracle Grid Engine is a powerful workload management tool for maximizing the business value of an organizations computing resources. By using Oracle Grid Engine, businesses can deliver their products faster, more efficiently, and with lower overall costs. This paper Introduces the Oracle Grid Engine product and some common and uncommon use cases.

Introduction to Workload Management

A computer is a great tool for doing work. For a desktop user that work looks like editing spreadsheets, visiting web sites, and playing Tetris when no one else is looking. The job of the computer is mostly just moving letters and numbers and windows around on the screen. For a server, however, the work looks very different. A server spends its days doing things like hosting web sites, processing email messages, and running calculations. Web sites and email are known as *services*. They're only useful if they're always (or almost always) there. If they move around or aren't always available, no one will use them. Services are most often run on dedicated machines.

Servers tend to be used for one of two purposes: running services or processing workloads. Services tend to be long-running and don't tend to move around much. Workloads, however, such as running calculations, are usually done in a more "on demand" fashion. When a user needs something, he tells the server, and the server does it. When it's done, it's done. For the most part it doesn't matter on which particular machine the calculations are run. All that matters is that the user can get the results. This kind of work is often called *batch*, *offline*, or *interactive* work. Sometimes batch work is called a *job*. Typical jobs include processing of accounting files, rendering images or movies, running simulations, processing input data, modeling chemical or mechanical interactions, and data mining. Many organizations have hundreds, thousands, or even tens of thousands of machines devoted to running jobs.

Now, the interesting thing about jobs is that (for the most part) if it's possible to run one job on one machine, it's possible to run 10 jobs on 10 machines or 100 jobs on 100 machines. In fact, with today's multi-core chips, it's often the case that it's possible to run 4, 8, or even 16 jobs on a single machine. Obviously, the more jobs it's possible to run in parallel, the faster it's possible get the work done. If one job takes 10 minutes on one machine, 100 jobs still only take ten minutes when run on 100 machines. That's much better than 1000 minutes to run those 100 jobs on a single machine. But there's a problem. It's easy for one person to run one job on one machine. It's still pretty easy to run 10 jobs on 10 machines. Running 1600 jobs on 100 machines is a tremendous amount of work. Now imagine that there are 1000 machines and 100 users all trying to running 1600 jobs each. Such an environment would be unmanageably complex.

To solve the problem of organizing a large number of jobs on a set of machines, workload managers were created. The role of a workload manager is to take a list of jobs to be executed and distributed them across the available machines. The workload manager makes life easier for the users because they don't have to track all their jobs themselves, and it makes life easier for the administrators because they don't have to manage users' use of the machines directly. It's also better for the organization in general because a workload manager will usually do a much better job of keeping the machines busy than users would on their own, resulting in much higher utilization of the machines. Higher utilization effectively means more compute power from the same set of machines, which means faster results and higher capacity without having to purchase additional resources.

A *cluster* is a group of machines cooperating to do some work. A workload manager and the machines it manages compose a cluster. A cluster is also often called a *grid*. There has historically been some debate about what exactly a grid is, but for most purposes *grid* can be used interchangeably with *cluster*. *Cloud*

computing is a hot topic that builds on concepts from grid/cluster computing. One of the defining characteristics of a *cloud* is the ability to "pay as you go." Grid Engine offers an accounting module that can track and report on fine grained usage of the system. Beyond that, Grid Engine now offers deep integration to other technologies commonly being used in the cloud, such as Apache Hadoop.

How Does Oracle Grid Engine Work?

An Grid Engine cluster is composed of execution machines, a master machine, and zero or more shadow master machines. The execution machines all run copies of the Grid Engine *execution daemon*. The master machine runs the Grid Engine *qmaster daemon*. The shadow master machines run the Grid Engine *shadow daemon*. In the event that the master machine fails, the shadow daemon on one of the shadow master machines will make that machine the new master machine. The qmaster daemon is the heart of the cluster, and without it no jobs can be submitted or scheduled. The execution daemons are the work horses of the cluster. Whenever a job is run, it's run by one of the execution daemons.

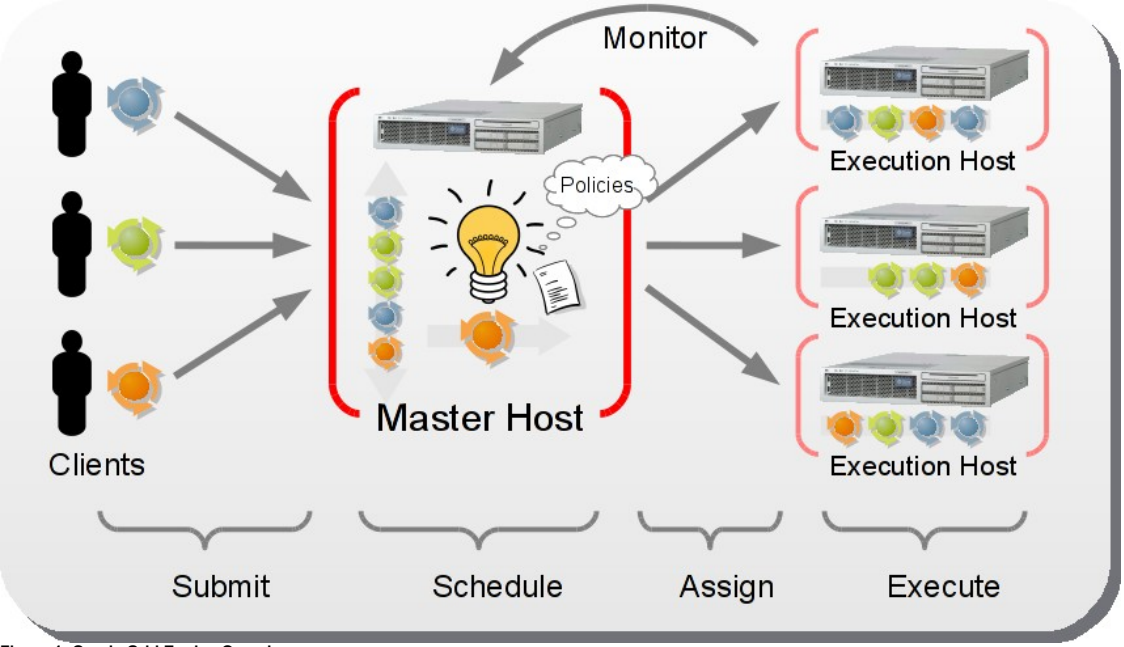

Figure 1: Oracle Grid Engine Overview

To submit a job to the cluster, a user uses one of the submission commands, such as *qsub*. Jobs can also be submitted from the graphical user interface, *qmon*. In the job submission the user includes all of the important information about the job, like what operation should be run, what kind of execution machine it needs, how much memory it will consume, how long it will run, etc. All of that information is then used by the qmaster to schedule and manage the job as it goes from pending to running to finished. For example, a qsub submission might look like: `qsub -wd /home/dant/blast -i /home/dant/seq.tbl -l mem_free=4G cross-blast.pl ddbdb`. This job searches for DNA sequences from the input file /home/dant/seq.tbl in the ddbdb sequence database. It requests

that it be run in the /home/dant/blast directory, that the /home/dant/seq.tbl file be piped to the job's standard input, and that it run on a machine that has at least 4GB of free memory.

Once a job has been submitted, it enters the pending state. On the next scheduling run, the qmaster will rank the job in importance versus the other pending jobs. The relative importance of a job is largely determined by the configured scheduling policies. Once the jobs have been ranked by importance, the most important jobs will be scheduled to available job *slots*. A *slot* is the capacity to run a job. Generally, the number of slots on an execution machine is set to equal the number of CPU cores the machine has; each core can run one job and hence represents one slot. Every available slot is filled with a pending job, if one is available. If a job requires a resource or a slot on a certain type of machine that isn't currently available, that job will be skipped over during that scheduling run.

Once the job has been scheduled to an execution machine, it is sent to the execution daemon on that machine to be run. The execution daemon executes the command specified by the job, and the job enters the running state. Once the job is running, it is allowed to continue running until it completes, fails, is terminated, or is requeued (in which case the prtocess starts over again). Along the way the job may be suspended, resumed, and/or checkpointed any number of times. (Grid Engine does not actually handle checkpointing itself. Instead, Grid Engine will trigger whatever checkpointing mechanism is available to a job, if any.)

After a job has completed or failed, the execution daemon cleans up after it and notifies the qmaster. The qmaster records the job's information in the accounting logs and drops the job from its list of active jobs. If the submission client was synchronous, the qmaster will notify the client that the job ended. Information about completed jobs is available through the *qacct* command-line tool or the Accounting and Reporting Console's web console.

In addition to traditional style batch jobs, as in the BLAST example above, Grid Engine can also manage interactive jobs, parallel jobs, and array jobs. An interactive job is like logging into a remote machine, except that Grid Engine decides to which machine to connect the user. While the user is logged in, Grid Engine monitors what the user is doing and recording the usage information in the accounting logs. A parallel job is a distributed job that runs across multiple nodes. Typically a parallel job relies on a parallel environment, like MPI, to manage its inter-process communication. An array job is similar to a parallel job except that it's processes don't communicate; they're all independent. Rendering an image is a classic array job example. The main difference between a parallel job and an array job is that a parallel job needs to have all of its processes running at the same time, whereas an array job doesn't; it could be run serially and would still work just fine.

Why Choose Oracle Grid Engine?

A number of different workload management solutions are available from a variety of different sources. What makes Grid Engine the best choice? At a very high level, the following are some of the main advantages of using Grid Engine:

- Scalability — Oracle Grid Engine is a **highly** scalable DRM system. Multiple customers are running clusters with thousands of machines, tens of thousands of CPU cores, and/or processing tens of millions of jobs per month.
- Flexibility — Grid Engine makes it possible to customize the system to exactly fit your needs.
- Advanced scheduler — Grid Engine does more than just spread jobs evenly around a group of machines. The Grid Engine qmaster supports a variety policies to fine-tune how jobs are distributed to the machines. Using the scheduling policies, you can configure Grid Engine to make its scheduling decisions match your organization's business rules.
- Reliability — Customers often report that Grid Engine just works and that keeps working. After the initial configuration, Grid Engine takes very little effort to maintain.

In addition, the Grid Engine software has a long list of features that make it a powerful, flexible, scalable, and ultimately useful workload management system.

Typical Use Cases

One of the easiest ways to understand Oracle Grid Engine is to see it in action. To that end, what follows are some typical use cases:

- Mentor Graphics, a leading EDA software vendor, uses the Grid Engine software to manage its regression tests. To test their software, they submit the tests as thousands of jobs to be run on the cluster. Grid Engine makes sure that every machine is busy running tests. When a machine completes a test run, Grid Engine assigns it another, until all of the tests are completed.
  In addition to using Grid Engine to manage the physical machines, they also use Grid Engine to manage their software licenses. When a test needs a software license to run, that need is reflected in the job submission. Grid Engine makes sure that no more licenses are used than are available.
  This customer has a diverse set of machines, including Solaris, Linux, and Windows. In a single cluster they process over 25 million jobs per month. That's roughly 10 jobs per second, 24/7. (In reality, their workload is bursty. At some times they may see more than 100 jobs per second, and at other times they may see less than 1.)
- Complete Genomics is using Grid Engine to manage the computations needed to do sequencing of the human genome. Their sequencing instruments are like self-contained robotic laboratories and require a tremendous amount of computing power and storage. Using Grid Engine as the driver for their computations, this customer intends to transform the way disease is studied, diagnosed and treated by enabling cost-effective comparisons of genomes from thousands of individuals. They currently have a moderate sized cluster, with a couple hundred machines, but they intend to grow that cluster by more than an order of magnitude.
- Rising Sun Pictures uses Grid Engine to orchestrate its video rendering process to create digital effects for blockbuster films. Each step in the rendering process is a job with a task for every frame. Grid Engine's workflow management abilities make sure that the rendering steps are performed in order for every frame as efficiently as possible.

- A leading mobile phone manufacturer runs a Grid Engine cluster to manage their product simulations. For example, they run drop test simulations with new phone designs using the Grid Engine cluster to improve the reliability of their phones. They also run simulations of new electronics designs through the Grid Engine cluster.
- D.E. Shaw is using Grid Engine to manage their financial calculations, including risk determination and market prediction. This company's core business runs through their Grid Engine cluster, so it has to just work. The IT team managing the cluster offers their users a 99% availability SLA.
  Also, this company uses many custom-developed financial applications. The configurability of the Grid Engine software has allowed them to integrate their applications into the cluster with little or no modifications.
- Another Wall Street financial firm is using a Grid Engine cluster to replace their home-grown workload manager. Their workload manager is written in Perl and was sufficient for a time. They have, however, now outgrown it and need a more scalable and robust solution. Unfortunately, all of their in-house applications are written to use their home-grown workload manager. Fortunately, Oracle Grid Engine offers a standardized API called DRMAA that is available in Perl (as well as C, Python, Ruby, and the Java™ platform). Through the Perl binding of DRMAA, this customer was able to slide the Grid Engine software underneath their home-grown workload manager. The net result is that the applications did not need to be modified to let the Grid Engine cluster take over managing their jobs.
- The Texas Advanced Computing Center at the University of Texas is #11 on the June 2010 Top500 list and uses Grid Engine to manage their 63,000-core Ranger cluster. With a single master managing roughly 4000 machines and over 3000 users working on over 1000 projects spread around throughout 48 of the 50 US states, the Ranger cluster weighs in as the largest Grid Engine cluster in production. Even though the cluster offers a tremendous amount of compute power to the users of the Teragrid research network (579 GigaFLOPS to be exact), the users and the Grid Engine master manage to keep the machines in the cluster at 99% utilization.
  The Ranger cluster is used by researchers around the country to run simulations and calculations for a variety of fields of study. One noteworthy group of users has run a 60,000-core parallel job on the Grid Engine cluster to do real-time face recognition in streaming video feeds.

Atypical Use Cases

One of the best ways to show Grid Engine's flexibility is to take a look a some unusual use cases. These are by no means exhaustive, but they should serve to give you an idea of what can be done with the Grid Engine software.

- A large automotive manufacturer uses their Grid Engine cluster in an interesting way. In addition to using it to process traditional batch jobs, they also use it to manage services. Service instances are submitted to the cluster as jobs. When additional service instances are needed, more jobs are submitted. When too many are running for the current workload, some of the service instances are

stopped. The Grid Engine cluster makes sure that the service instances are assigned to the most appropriate machines at the time.

- One of the more interesting configuration techniques for Grid Engine is called a *transfer queue*. A transfer queue is a queue that, instead of processing jobs itself, actually forwards the jobs on to another service, such as another Grid Engine cluster or some other service. Because the Grid Engine software allows you to configure how every aspect of a job's life cycle is managed, the behavior around starting, stopping, suspending, and resuming a job can be altered arbitrarily, such as by sending jobs off to another service to process.

- A Grid Engine cluster is great for traditional batch and parallel applications, but how can one use it with an application server cluster? There are actually two answers, and both have been prototyped as proofs of concept.

  The first approach is to submit the application server instances as jobs to the Grid Engine cluster. The Grid Engine cluster can be configured to handle updating the load balancer automatically as part of the process of starting the application server instance. The Grid Engine cluster can also be configured to monitor the application server cluster for key performance indicators (KPIs), and it can even respond to changes in the KPIs by starting additional or stopping extra application server instances.

  The second approach is to use the Grid Engine cluster to do work on behalf of the application server cluster. If the applications being hosted by the application servers need to execute longer-running calculations, those calculations can be sent to the Grid Engine cluster, reducing the load on the application servers. Because of the overhead associated with submitting, scheduling, and launching a job, this technique is best applied to workloads that take at least several seconds to run. This technique is also applicable beyond just application servers, such as with SunRay Virtual Desktop Infrastructure.

- A research group at a Canadian university uses Grid Engine in conjunction with the open source Cobbler project to do automated machine profile management. Cobbler allows a machine to be rapidly reprovisioned to a pre-configured profile. By integrating Cobbler into their Grid Engine cluster, they are able to have Grid Engine reprovision machines on demand to meet the needs of pending jobs. If a pending job needs a machine profile that isn't currently available, Grid Engine will pick one of the available machines and use Cobbler to reprovision it into the desired profile.

  A similar effect can be achieved through virtual machines. Because Grid Engine allows jobs' life cycles to be flexibly managed, a queue could be configured that starts all jobs in virtual machines. Aside from always having the right OS profile available, jobs started in virtual machines are easy to checkpoint and migrate.

- With the 6.2 update 5 release of the Grid Engine software, Grid Engine can manage Apache Hadoop workloads. In order to do that effectively, the qmaster must be aware of data locality in the Hadoop HDFS. The same principle can the applied to other data repository types such that the Grid Engine cluster can direct jobs (or even data disguised as a job) to the machine that is closest (in network terms) to the appropriate repository.

- One of the strong points of the Grid Engine software is the flexible resource model. In a typical cluster, jobs are scheduled against things like CPU availability, memory availability, system load, license availability, etc. Because the Grid Engine resource model is so flexible, however, any number of custom scheduling and resource management schemes are possible. For example, network bandwidth could be modeled as a resource. When a job requests a given bandwidth, it would only be scheduled on machines that can provide that bandwidth. The cluster could even be configured such that if a job lands on a resource that provides higher bandwidth than the job requires, the bandwidth could be limited to the requested value (such as through the Solaris Resource Manager).

Conclusion

Oracle Grid Engine is a powerful and flexible workload management solution capable of addressing a broad range of problems across a number of industry verticals. It's ability to scale to large cluster sizes and high throughput workloads, coupled with its ease of application integration, makes it an effective solution in many workload and resource management situations.

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

White Paper Title
[Month] 2010
Author: [OPTIONAL]
Contributing Authors: [OPTIONAL]

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

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 and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. UNIX is a registered trademark licensed through X/Open Company, Ltd. 0110