An Oracle White Paper
August 2010

Extreme Scalability Using Oracle Grid Engine Software: Managing Extreme Workloads

ORACLE

Executive Overview

As user data sets and computing needs increase in size and complexity, administrators are faced with building larger and more powerful clusters to keep pace. Some of the common practices in cluster administration fail to scale, and break down as cluster sizes grow. This white paper explains several best practices for building super-scale clusters that utilize Oracle™ Grid Engine software, and reveals other practices to avoid.

Introduction

As the problems being tackled by researchers and enterprises grow in complexity, the tools being used also must grow in power. At the leading edge of the line-up of power tools in a researcher's arsenal are super-scale compute clusters. With teraFLOPS of compute power, petabytes of storage, and terabytes of memory, these computing giants make possible computations and simulations that would otherwise take orders of magnitude longer to run or be out of reach altogether. Future requirements are beginning to emerge that are expected to require a three orders of magnitude increase in computing speed and associated infrastructure.

The art of building and running a super-scale compute cluster is complex and challenging. When growing a cluster into the super-scale range, issues that are minor or go unnoticed in a smaller cluster suddenly become glaring issues. What works at small scale offers no promise of working at super-scale levels.

The Oracle Grid Engine software is a distributed resource management (DRM) system that is commonly used in compute clusters to manage the distribution of workloads to available resources. The software is popular not only among the small and medium clusters common in many organizations, but also in larger clusters — even super-scale compute clusters. In these larger clusters, configuring the Oracle Grid Engine software requires an understanding of the way the software works and how to extract maximum performance.

This white paper provides information to help administrators understand more about the inner workings of the Oracle Grid Engine software, and how to tune it to get the maximum effect in large-scale and super-scale environments. While this information applies mostly to larger clusters, it also can be useful for administering any size Oracle Grid Engine cluster. This paper discusses the following topics:

- Chapter 1, “Super-Scale Computing,” introduces super-scale computing.
- Chapter 2, “Typical Challenges,” describes some of the common issues faced with super-scale compute clusters.
- Chapter 3, “The Role of Oracle Grid Engine Software,” explains what the Oracle Grid Engine software brings to a super-scale compute cluster.
- Chapter 4, “Best Practices,” presents several suggestions for architecting and configuring a super-scale Oracle Grid Engine cluster.

This white paper is intended for administrators interested in the issues around scaling an Oracle Grid Engine cluster into the super-scale range. It may also be useful for Oracle Grid Engine administrators interesting in performance and scalability in general.

Chapter 1: Super-Scale Computing

What is Super-Scale Computing?

Super-scale computing is the application of very large compute clusters to the task of solving computational problems. A compute cluster is a collection of computing resources that work together to solve a problem or a set of problems. The cluster usually is federated through some sort of software middleware, such as a parallel library such as the Message Passing Interface (MPI), a distributed resource manager (DRM) such as Oracle Grid Engine, or an application framework like Apache Hadoop.

In typical applications, a compute cluster can range in size from a few nodes to a few hundred nodes. For most application needs, compute clusters in the range of tens to hundreds of nodes are sufficient. However, the size of the data or the scale of the computations for some applications is such that significantly more compute power is required. These applications need super-scale compute clusters with hundreds of nodes and thousands of cores. Such massive compute clusters are expensive to build and maintain, but provide a level of computational horsepower found nowhere else in the world.

Published twice a year, the Top500 Supercomputer Sites is a list of the most powerful systems. As of June 2009, the world's fastest super-scale compute cluster is the Roadrunner system installed at the U.S. Department of Defense (Figure 2-1). The cluster is composed of 278 refrigerator-sized racks containing 18,802 processors with a total of 122,400 cores that occupies 5,200 square feet[1]. The systems are connected by over 55 miles of fiber optic cable, using both InfiniBand and Gigabit Ethernet technology. The entire cluster—including cables—weighs over 500,000 lbs, consumes 2.35 Megawatts of power, and has a maximum computing potential of over 1 petaFLOP (over 1,000,000,000,000,000 operations per second). For comparison, a modern desktop workstation with two quad-core Intel® processors offers approximately 100 gigaFLOPS of peak performance.

The precise definition of super-scale is still open to debate. The general consensus is that a super-scale cluster contains more than 1,000 nodes or more than 10,000 cores, or delivers more than 100 teraFLOPS of performance.

[1] See http://www.top500.org/system/9485.

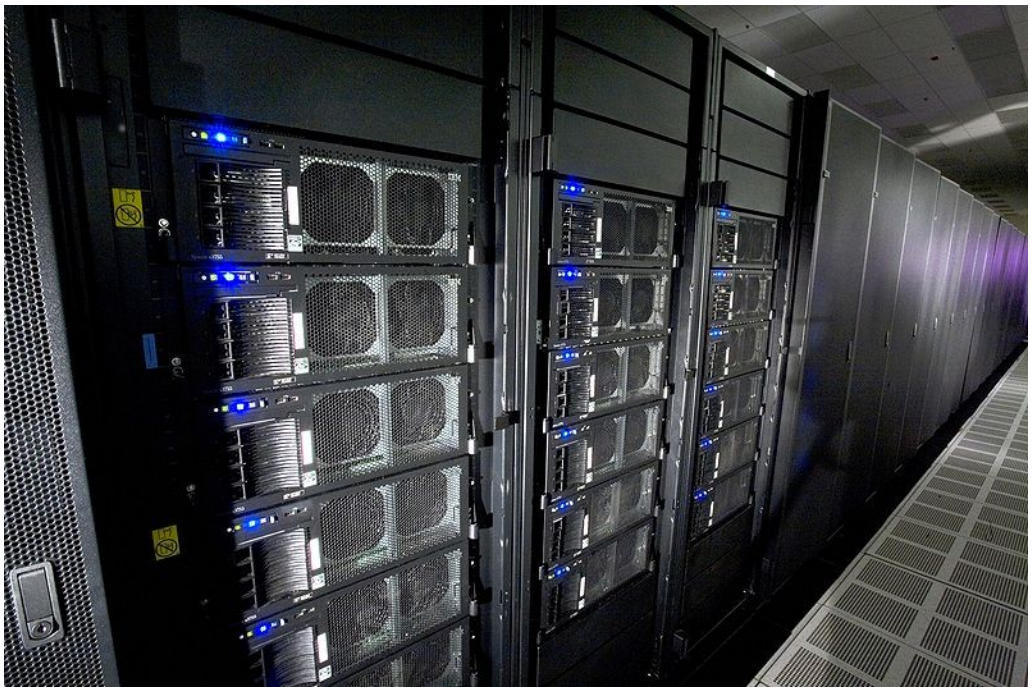

Figure 2-1. The National Nuclear Security Administration's Roadrunner Cluster[2].

Supporting text should contain benefit/solution information such as which business problems exist and how they are solved, the ROI/value produced by addressing the problem, and which solution(s) or pieces of the solution Oracle provides.

Who is Using Super-Scale Computing?

Building a super-scale compute cluster can be a costly affair. The Roadrunner cluster shown in Figure 2-1 cost approximately $100,000,000 to build. While many super-scale compute clusters are not as expensive to assemble, they do require a significant investment and commitment from the purchasing organization.

Because of the costs and challenges involved, today's super-scale compute clusters tend to be limited to organizations with large computational needs and commensurately large budgets. The classic users of super-scale compute clusters are research organizations and government funded laboratories. For example, the U.S. Department of Energy, the U.S. Department of Defense, and the National Aeronautics and Space Administration (NASA) all have systems on the Top500 List. In addition, Internet information

[2] Photo courtesy Los Alamos National Laboratory. Unless otherwise indicated, this information has been authored by an employee or employees of the University of California, operator of the Los Alamos National Laboratory under Contract No. W-7405-ENG-36 with the U.S. Department of Energy. The U.S. Government has rights to use, reproduce, and distribute this information. The public may copy and use this information without charge, provided that this Notice and any statement of authorship are reproduced on all copies. Neither the Government nor the University makes any warranty, express or implied, or assumes any liability or responsibility for the use of this information.

brokers such as a Google, Yahoo!, and Facebook, also use super-scale compute clusters. However, the nature of the workload leads information brokers to use massive numbers of less reliable, lower cost systems to help mitigate costs.

How are Super-Scale Clusters Being Used?

The most common use for a super-scale compute cluster is research, from high-energy physics to meteorology and astronomy. The largest clusters on the Top500 List are used by researchers to perform calculations and process data that would otherwise take weeks, months, or even years to do on less grandiose computing resources.

The Texas Advanced Computing Center's Ranger cluster is a good example. Built through a $30,000,000 grant from the National Science Foundation in 2007, the cluster was targeted to be the world's fastest supercomputer at over a half petaFLOP in peak, theoretical performance when it launched. In the June, 2009 Top500 List, the Ranger cluster was listed at number eight. The cluster is part of the Teragrid, a collection of computational resources distributed across the U.S. through member organizations. (When it came online, the Ranger system was more powerful than all of the other Teragrid systems combined.)

Researchers with permission to use the Teragrid can leverage its enormous compute power to answer open questions in their research. One notable use of the Ranger cluster through the Teragrid has been the Pacific Northwest National Laboratory project on video facial recognition. Ron Farber and his team created an application that ran distributed across 60,000 CPU cores on the Ranger cluster to perform real-time face recognition in videos with 99.8% accuracy. Because of the sheer volume of video data being created every moment, the enormous scale of the Ranger cluster was important for testing the scalability of the recognition algorithms.

Figure 2-2. The Ranger cluster at the Texas Advanced Computing Center[3].

Another example is the EINSTEIN cluster at the Naval Oceanographic Office Major Shared Resource Center (NAVO MSRC), which debuted at number 26 on the Top500 List with over 12,000 processors. The NAVO MSRC is best known for the Navy Operations Global Atmospheric Prediction System (NOGAPS), which produced the most accurate five-day prediction of Hurricane Katrina's behavior. Today, the EINSTEIN system is used for modeling global weather patterns for prediction.

A popular commercial super-scale compute cluster example is the Google compute cluster. While details are confidential, it is estimated that Google uses approximately 450,000 production servers that are distributed across at least a dozen datacenters throughout the world. Instead of simulation or modeling, Google's compute clusters are the engine behind the popular search tool. They also are used to power offline data mining jobs against the many terabytes of data Google has archived.

Examples Uses for Large-scale Clusters

[3] Photo courtesy the Texas Advanced Computing Center. See http://www.tacc.utexas.edu/resources/hpc/

A variety of disciplines can take advantage of large-scale compute clusters. In the examples below, the applications themselves are not highly scalable—but thousands of smaller jobs must be scheduled, resources allocated, and the execution managed.

- *Image comparisons*

Image recognition is gaining in popularity, and poses interesting challenges for compute clusters. For example, consider a large database of images that need to be compared. Some kind of image processing algorithm must be used. If each application is single-threaded, then thousands of jobs would need to be submitted into queues. With potentially millions of images needing to be compared to thousands of source images, it is imperative that the DRM software be able to scale and handle these massive loads.

- *Image rendering*

The software rendering of images, particularly those seen at the movies or on TV, can be an enormous computational and scheduling challenge. For example, a two hour long movie requires 216,000 frames to be rendered (30 frames/sec x 60 sec/minute x 60 minutes/hour x 2 hours), with each frame undergoing multiple rendering phases. With so many frames needing processing, some kind of job management for final rendering is critical.

- *Molecular modeling*

Simulating millions of compounds and analyzing how they react with other compounds requires enormous computing resources. Since many of these jobs can be run at the same time, and independently of one another, compute resource management is critical for both throughput and cluster utilization.

Chapter 2: Typical Challenges

One reason the clusters that top the Top500 List are held in such reverence is that building, maintaining, and efficiently employing a super-scale compute cluster is no small feat. A cascading series of challenges must be overcome before such a cluster can be considered a success.

- *Finding money*

The first challenge is cost. In some cases, costs are covered by a grant from a government agency. However, winning the grant is itself a long and challenging process. In other cases, the cost of building the cluster is borne internally, which can be a process equally as long and challenging.

- *Building a facility*

Once there is sufficient funding to build and run a super-scale compute cluster, the next challenge is finding or building a facility to house the cluster. With literally miles of cables and tons of computing hardware sometimes filling a space the size of a football field or more, the physical logistics of housing a super-scale compute cluster can be very daunting—especially when considering cluster power and cooling demands. In many cases, organizations are as proud of the datacenter housing the cluster as they are of the cluster itself.

- *Making effective use of the cluster*

Once a super-scale compute cluster is up and running, the next challenge is making effective use of it. Software scalability often is a limiting factor. The fact that a software application takes 100 minutes when run across 10 machines does not mean that running the same application across 1,000 machines will make it complete in one minute. In fact, researchers and programmers often discover that once an application's maximum scalability is reached, adding additional compute resources can increase the time it takes for the application to run to completion. An application that takes 100 minutes on 10 machines might take 200 minutes when run on 1,000 machines.

As more compute resources are added to a distributed application, more and more time is spent just communicating among the distributed tasks. Writing distributed applications that can make effective use of a super-scale compute cluster is a delicate and challenging business. Unfortunately, most users are not career computer scientists or programmers. In fact, most are career physicists, chemists, or mathematicians who have become self-taught computer programmers in the course of trying to complete their research. The challenge of writing super-scalable software applications is significant, even for a skilled computer scientist. For a self-taught researcher, it can be nearly insurmountable.

Chapter 3: The Role of Oracle Grid Engine Software

If it's so difficult to write super-scale software applications, then why not write regular software applications and only run them on a subset of the compute cluster? In fact, most users of compute clusters take this very approach. Only in very small compute clusters or the rare case where scalability is a central focus of the project, such as the previously mentioned video facial recognition application, does a single application consume all of a compute cluster's resources. Instead, applications share the compute cluster's resources, each using as much of the cluster as is allotted to it and can be efficiently used by the application.

If multiple applications being operated by multiple users and groups are to effectively share a compute cluster, a central entity is needed to coordinate the resource allocation. Such an entity usually is known as a distributed resource management (DRM) system. A variety of commercial and open-source DRM systems are available. Leading commercial DRM solutions include the Oracle Grid Engine software, Platform LSF, and Altair PBS Pro products. Popular open-source DRM solutions include Oracle's Grid Engine project, the Torque open source project, and the University of Wisconsin's Condor project.

Oracle Grid Engine Software Overview

The Oracle Grid Engine software is a distributed resource management system that fosters increased utilization, better workload throughput, and higher end-user productivity by harnessing existing compute resources. By transparently selecting the resources that are best suited for each segment of work, the Oracle Grid Engine software is able to distribute workloads efficiently across a resource pool while shielding end users from the inner workings of the compute cluster.

In order to facilitate the most effective and efficient scheduling of workloads to available resources, the Oracle Grid Engine software gives administrators the ability to accurately model as a resource any aspect of the compute environment that can be measured, calculated, specified, or derived. The software can monitor and manage resources that are concrete, such as CPU cores or system memory, as well as abstract resources, such as application licenses or mounted file systems. In addition to allocating resources based on workload requirements, the Oracle Grid Engine software provides an arsenal of advanced scheduling features that allow administrators to model not only the compute environment, but also the business rules that control how the resources in that compute environment should be allocated.

The architecture of an Oracle Grid Engine cluster is fairly straightforward (Figure 3-1). A central master component, known as the QMaster, coordinates communications for the cluster and is responsible for scheduling workloads on available resources. Optional Shadow Daemons monitor the state of the QMaster and can start a new QMaster if the current master process or node becomes unresponsive. On each compute resource, an Execution Daemon manages the execution of workloads on the node. Users interact with the cluster via command-line tools, graphical user interfaces, and Distributed Resource Management Application API (DRMAA)-enabled applications. All activity in the cluster is logged into an accounting database that can be accessed by users and administrators through the Accounting and Reporting Console (ARCo).

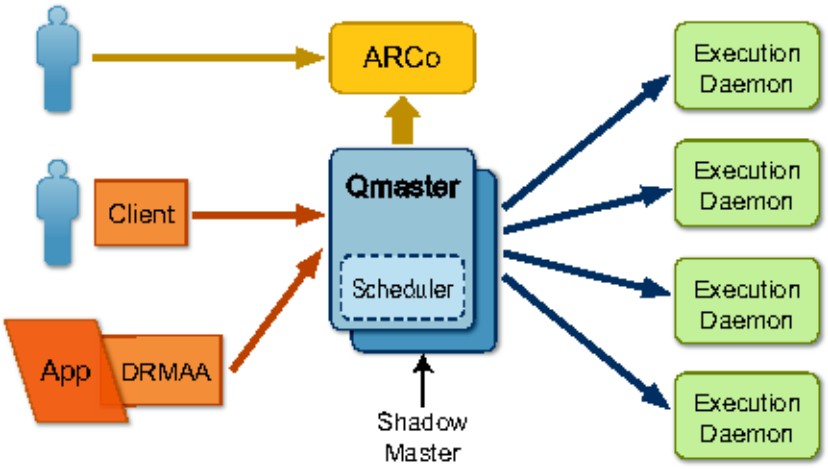

Figure 3-1. The Oracle Grid Engine software architecture.

Oracle Grid Engine software includes the Service Domain Manager, software that enables the sharing of resources among separate clusters (Figure 3-2). Through the Service Domain Manager software, an administrator can configure service-level objectives to govern service levels in the managed clusters. As load in one cluster increases beyond the boundaries set by the service-level objectives, resources from less heavily burdened clusters can be automatically redeployed to the overloaded cluster until the overage has passed.

Should no free resources be available locally, the Service Domain Manager software also has the ability to provision resources from a compute cloud provider, such as Amazon's Elastic Compute Cloud (EC2), to add to an overloaded cluster. When the overage ends, these resources are returned automatically to the cloud. In addition, the Service Domain Manager software is able to remove unused resources from managed clusters and place them in a spare pool of resources. Resources in this spare pool optionally can have power management applied to reduce a cluster's overall power consumption during off-peak periods.

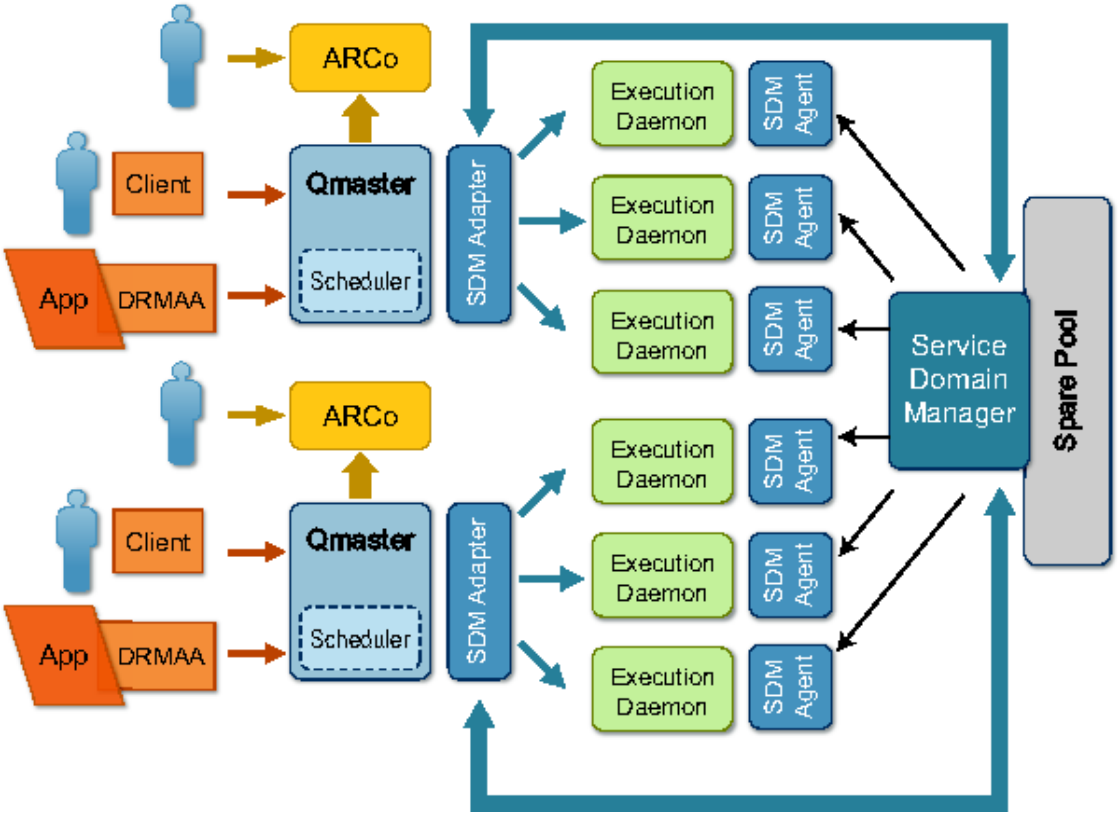

Figure 3-2. The Service Domain Manager.

Why Use Oracle Grid Engine?

The Oracle Grid Engine software is a highly scalable and robust DRM system that offers advanced scheduling features and support for open standards. Key features of the Oracle Grid Engine software include:

- Fair-share scheduling
- Functional ticket scheduling
- Advance reservation of resources
- Dynamic resource reservation to prevent the starvation of jobs with large resource needs
- A highly extensible resource model
- Fine-grained resource quotas
- Job submission verification
- C and Java™ programming language binding implementations of the Open Grid Forum's Distributed Resource Management Application API (DRMAA)

- Scriptable interfaces with optional XML output
- Advanced accounting tools
- Real-time cluster monitoring tools
- A flexible plug-in mechanism for integrating a variety of parallel programming environments, such as MPI, and checkpointing frameworks

The Oracle Grid Engine software also offers industry-leading scalability. The Texas Advanced Computing Center's Ranger system uses the Oracle Grid Engine cluster to manage the workload on their 63,000 core, 580 TeraFLOP compute cluster— deployed as a single Oracle Grid Engine cluster. The Oracle Grid Engine software is also known for its reliability. It is not uncommon for an Oracle Grid Engine cluster to run for years without the need for remedial attention. Additionally, the software is designed to foster maximum flexibility. Most behaviors in the software can be configured or overridden with administrator-defined scripts. The ability to configure the Oracle Grid Engine software to meet such a wide variety of needs has helped to make it one of the world's most widely deployed DRM systems.

Based on the Grid Engine open-source project, Oracle Grid Engine software has a strong and vibrant community. The Grid Engine open-source community members actively share tips, best practices, and assistance through the community mailing lists and wiki. The Grid Engine open source community offers Grid Engine and Oracle Grid Engine users the opportunity to connect with other users and the Oracle Grid Engine development team to find answers to questions and help with problems. Because the community is so active and strong, it is often cited as one of the Oracle Grid Engine software's most important features.

Chapter 4: Best Practices

Building highly-scalable software applications is difficult — and so is building and configuring a large-scale or super-scale compute cluster and the DRM software that manages the workload. Many practices that work well on a smaller scale cease to be viable as the system count reaches into the hundreds or thousands. This chapter provides best practices for building and configuring large-scale compute clusters.

Architectural Considerations

When architecting a large-scale compute cluster, some basic best practices apply.

Master Host

In clusters that see significant throughput, the node that hosts the `qmaster` process should not also be used to host an execution daemon. When the pending job list is long, the scheduling process can be quite compute-intensive. Having running jobs competing with the scheduler for CPU time can introduce unnecessary job latency.

The Oracle Grid Engine `qmaster` process has been multithreaded since the 6.0 release. As of the 6.2 release, the scheduler process also is just another thread in the `qmaster` process. In a typical `qmaster` process, several threads play key roles. Two threads accept incoming job submissions and administrative requests. Another two threads process the accepted job submissions and administrative requests. One thread runs the scheduler, and a variety of other activities are handled by a number of other threads. In total, a typical system running Oracle Grid Engine software 6.2 update 3 `qmaster` uses approximately 13 active threads.

The `qmaster` process profits significantly when two processor cores are available. Doubling the core count to four processor cores results in a noticeable performance increase, but not as significant an improvement as going from one processor core to two cores. Increasing the number of processor cores available to the `qmaster` process beyond four cores does not produce a significant performance improvement.

Several considerations are important when sizing the memory requirements for the machine running the `qmaster` process: the number of jobs to be handled by system at any one time (running plus pending), the number of hosts in the cluster, and the number of queues in the cluster. Key calculations include:

- Multiply the maximum number of jobs expected by 50 KB to obtain a rough estimate of the amount of memory required by the `qmaster` process for jobs.
- Multiply the number of queues by the number of hosts by 100 KB to get a rough estimate of the amount of memory needed by the `qmaster` process for host and queue information.
- Add the two results together to get a rough estimate of the total memory needs of the `qmaster` process.

For example, a `qmaster` process that is expected to handle a peak load of 20,000 active jobs requires approximately 1 GB of memory for job information. If the `qmaster` process is managing a cluster of 1,000 hosts in five queues, it needs approximately an additional 500 MB of memory for host and queue information. The total memory needs approach 1.5 GB.

Networking

Inter-process communication in an Oracle Grid Engine cluster is minimal, with inter-daemon communications consisting of:

- Messages between clients (running on submission hosts or execution hosts) and the `qmaster` process
- Job execution orders (potentially containing the submitted job scripts) from the `qmaster` process to execution daemons
- Load reports from execution daemons to the `qmaster` process.

The amount of traffic produced by the client messages and job execution orders is governed by the rate of job submission and scheduling. The amount of traffic produced by the load reports is governed by the load report interval and the number of resources whose values have changed during the preceding load report interval. In the majority of clusters, the network traffic produced by inter-process communications by the Oracle Grid Engine software does not require a special high-speed network interconnect (Figure 4-1). On the other hand, the network interconnect among the execution hosts, and between the execution hosts and the data source(s), often requires a special high-speed interconnect. However, the amount of network bandwidth required is governed by the network profile of the workload the cluster is executing. It is unrelated to the Oracle Grid Engine cluster.

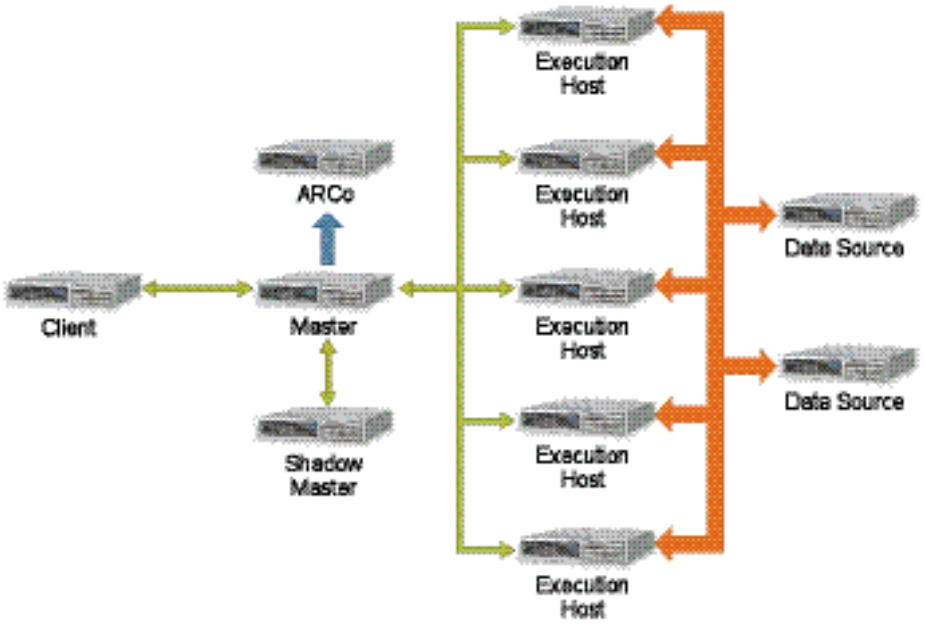

Figure 4-1. Network bandwidth requirements vary in Oracle Grid Engine clusters.

Cluster Configuration

One of the hallmarks of the Oracle Grid Engine software is its flexibility. The software can be configured and tuned for optimal management of a wide variety of workloads and cluster sizes and types. When configuring the Oracle Grid Engine software for managing a large-scale cluster, some well-known best practices apply.

Load Reports

The execution daemons send periodic load reports to the `qmaster` process. These reports contain the values of all resources that have changed since the last time the execution daemon sent a load report. The frequency of the load reports is set by the `load_report_interval` attribute of the host configuration.

By default, the load report interval is set to 40 seconds. Setting the load report interval to a shorter value means that the information used by the scheduler to make scheduling decisions is updated more frequently. It also means that the `qmaster` process must process more load reports in a given time interval, which often is an acceptable trade-off in a smaller cluster. Setting the load report to a larger value means that the resource information used by the scheduler is updated less frequently, possibly resulting in less accurate scheduling decisions. It also reduces the burden on the `qmaster` process, since there are fewer load reports to process.

Prior to the Oracle Grid Engine 6.2 release, load reports contained the values of all resources, regardless of whether they had changed. In large clusters, the load report interval was a significant factor when tuning the cluster. For example, the TACC Ranger cluster used a load report interval of five minutes prior to upgrading to the Oracle Grid Engine 6.2 release family.

For clusters running Oracle Grid Engine software release 6.2 or later, the load report interval is a less significant factor for scalability. If the `qmaster` process in a very large cluster consumes large amounts of CPU time even when the job submission rate is low, setting the load report interval to a larger number can provide some relief. If the `qmaster` process is located on a host running the Solaris™ Operating System or OpenSolaris™ operating environment, the Solaris Dynamic Tracing facility (DTrace) script included with the Oracle Grid Engine software can be used to look at the number of reports being received by the `qmaster` process in a given interval. The #rep field in the output of the DTrace script indicates the number of load reports received by the `qmaster` process during the script polling period. Note that the script polling interval is not related to the cluster's load report interval or scheduling interval, and can be configured through command-line options when running the script.

Scheduling Interval

In some cases it is undesirable for the scheduler to run constantly, as it would needlessly consume CPU time on the master host. For this reason, the scheduler runs only periodically, at a frequency determined by the `scheduling_interval` attribute in the scheduler configuration. In addition, a scheduling run can optionally be triggered by job submission and/or job completion. The default settings cause a scheduling run to happen every 15 seconds, but not in response to job submission or completion.

The first thing to consider when configuring the scheduling interval is the system on which the `qmaster` process is running. On a single-core machine, the scheduling interval is very important. When only one processor is available, the scheduler must compete with the other threads in the `qmaster` process for CPU time. On machines where two or more cores are available to the `qmaster` process, the scheduler does not compete with other `qmaster` process threads for compute time, and is free to consume up to 100% of a processor core. The `qmaster` process operates at peak capacity when scheduling runs happen back-to-back. Note that peak capacity does not mean peak efficiency. Since the scheduler's CPU time is effectively free, it is better to use every bit of capacity, regardless of efficiency.

Scheduling Runs

In clusters with very high job throughput, the duration of the scheduler runs can become a performance issue, especially when advanced policies are in use. The scheduler reviews the resource needs of every job waiting to be scheduled and matches those needs to available hosts in open queues. The more resources, hosts, and queues in the cluster, the more options the scheduler must consider and the longer it takes to make a scheduling decision. To minimize the amount of scheduler overhead, it is highly recommended in large clusters to avoid adding unnecessary configuration objects, such as additional queues, resources, and projects. It is important to give careful consideration to configuration decisions so as not to over-engineer the solution, resulting in needless scheduler overhead.

If advanced policies are in use, the entire list of pending jobs is reordered to reflect the effect of the active policies before the scheduler assigns the first job. This may result in a priority calculation for every job. Limiting the number of configuration objects in use is one way to reduce the cost of these priority calculations. Another way to limit the amount of compute time consumed by calculating job priorities is to use the `max_functional_jobs_to_schedule` attribute in the scheduler configuration. The `max_functional_jobs_to_schedule` attribute limits the number of jobs reprioritized during a scheduling run by the functional ticket policy. By default, the attribute is set to 200, indicating that only the first 200 jobs are considered by the functional policy. Note that this attribute only applies to the functional ticket policy. Other ticket-based policies, such as the share tree policy and the override policy, are unaffected by this attribute.

Another classic technique for reducing scheduling overhead is to combine multiple batch jobs into a single array job. This technique reduces the work the scheduler has to do — it can handle multiple jobs as a single job and perform priority calculations only once. The technique is very effective, and is widely used in production sites. The main limitation to this technique is that the advanced policies also treat this composite job like a single job, potentially resulting in inappropriate priority assignment to some tasks.

Execution Host Spool Directories

When the `qmaster` process sends a job to an execution daemon to be executed, the execution daemon spools that job's information to a local disk so that it can be recovered in the event of a failure. The spool location is determined by the `execd_spool_dir` attribute in the host configuration for the host or inherited from the global host configuration.

It is recommended that the spool directory for an execution daemon reside on a local file system. The default setting is to place the spool directory under the cluster's cell directory, which most often resides on a shared file system. Placing the execution daemon's spool directory on a shared file system increases the time required to write and then read the job information. This increases the load on the file system as well as the amount of network traffic, and does not result in tangible benefit. Placing the spool directory for an execution daemon on a file system that is local to the execution host avoids these problems without loss of flexibility or functionality.

Master Host Spooling Method

During the Oracle Grid Engine `qmaster` installation process, the installer asks the administrator to specify the type of spooling to use for the `qmaster` process. The choices are `classic` or `berkeley`. The right answer depends on the size and availability needs of the cluster.

- *Classic spooling*

Classic spooling stores `qmaster` process data in flat files in the `qmaster` process spool directory. One advantage to this approach is that it is easier to diagnose problems. It also works well with shared file systems. However, classic spooling can be a performance bottleneck, especially on shared file systems. When using classic spooling, the `qmaster` process creates, reads, and removes small files at a high frequency. When the `qmaster` process spool directory is hosted on a shared file system, every file operation incurs the additional penalty of the network latency and the shared file system protocol overhead. When hosted locally on a high-performance file system, however, classic spooling can be a reasonable approach. For example, classic spooling on the Solaris ZFS™ file system has been shown to be at least as fast as local `berkeley` spooling in micro-benchmarks.

- *Berkeley spooling*

The `berkeley` spooling method stores `qmaster` process data in a local embedded Berkeley Database or in a remote Berkeley Database server. The advantage of `berkeley` spooling is that it is fast. However, `berkeley` spooling can make diagnosing problems more difficult, and it does not work with NFS versions 2 or 3. It is recommended that `berkeley` spooling be used in large-scale clusters that do not use high availability. The loss in ease of diagnosis is outweighed by the gain in performance.

When configuring an Oracle Grid Engine cluster to use shadow daemons, the native high availability option, the cluster's cell directory and `qmaster` process spool directory must be made available via a shared file system or a highly synchronized mirror. Because local `berkeley` spooling does not work with NFS versions 2 or 3, the `qmaster` process spool directory must be shared via NFS version 4 or through another shared file system technology, such as the Lustre™ file system or Panasas parallel file system technologies.

The alternative to local `berkeley` spooling over a shared file system is to use `remote berkeley` spooling to a Berkeley Database server. This approach has two main disadvantages. First, it offers notably lower performance than local `berkeley` spooling. Second, the Berkeley Database server is a single point of failure, requiring its own high availability solution. For these reasons, classic spooling is

often selected for high availability clusters, even though the performance penalty for classic spooling over a shared file system is noticeable.

File System Sharing

In an Oracle Grid Engine cluster, it is assumed that the cluster's common directory, contained in the cluster's cell directory, is available via a shared file system. It must reside on a shared file system in a highly available cluster. For convenience, the entire Oracle Grid Engine root directory often is shared in small clusters. The root directory contains the cell directory, the `qmaster` process spool directory, and the product binaries. This practice makes installing execution hosts extremely easy, as all of the required software is available via the shared file system. The only thing that needs to be installed locally on the execution hosts is the init script for the execution daemon, which is installed by the execution daemon installation script.

In a large cluster, this practice can be detrimental. In clusters that are not highly available, placing the `qmaster` process (classic) spool directory on a shared file system can negatively impact performance. When running parallel jobs, accessing the product binaries via a shared file system can have a severely negative effect. For every slave task in a parallel job, the execution daemon running the task launches a shepherd process. The binary for the shepherd process is located with the rest of the product binaries. When a parallel job starts, the job's slave tasks usually all start at approximately the same time. If a parallel job is large, that could mean hundreds, thousands, or even tens of thousands of execution daemons trying to start shepherd processes simultaneously. If the shepherd binary is being accessed via a shared file system, all of the execution hosts try to access the shared file system at the same time, causing a large spike in load and potentially overwhelming the shared file system server.

For this reason, it is always better from a performance perspective to have a local copy of the product binaries on every execution host. In a large-scale cluster, this practice can result in significant administrative overhead. However, if a successful software deployment strategy is already in place, the added cost tends to be minimal.

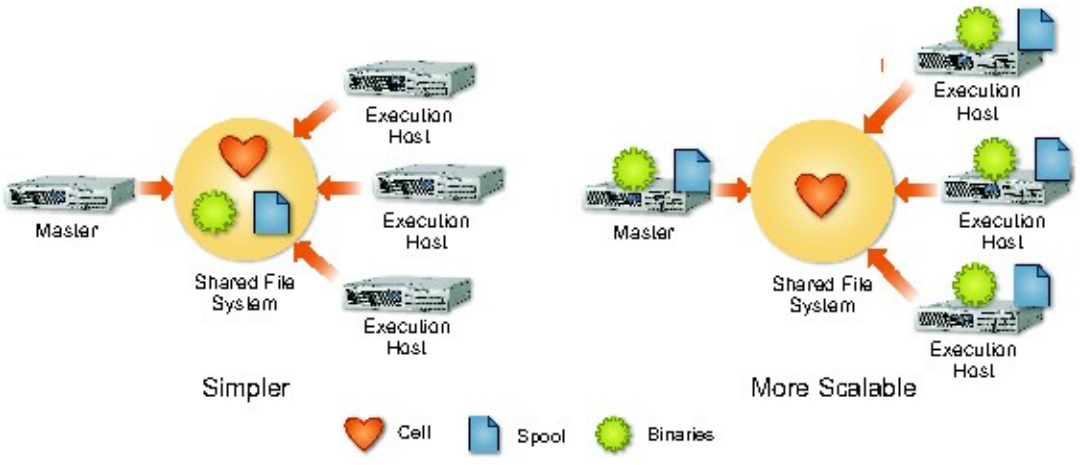

Figure 4-2. Keeping a local copy of product binaries on every execution host can help improve performance.

Portable Data Collector

The Portable Data Collector (PDC) is a module in the execution daemon that is responsible for gathering useful information about running jobs and enforcing CPU time and memory limits. By default, the PDC runs once per second in order to gather complete and accurate information and promptly enforce limits. When the PDC runs, it can produce a noticeable amount of CPU load on hosts running a Linux operating system. In clusters with extremely compute-intensive workloads, this amount of CPU load can be unacceptable.

The frequency with which the PDC runs can be controlled by the `PDC_INTERVAL` setting for the `execd_params` attribute in the global host configuration. It can be set to a number, indicating the number of seconds between PDC runs. The default setting is 1. The `PDC_INTERVAL` also can be set to the value `PER_LOAD_REPORT`. Doing so ensures the PDC is run immediately before the execution daemon builds each load report, which happens every 40 seconds by default. (See the Load Reports section earlier in this chapter.) The PDC can be disabled by setting the `PDC_INTERVAL` to `NEVER`.

Setting the PDC to run less frequently reduces CPU consumption by the execution daemons, leaving more compute time available for execution jobs. Since the PDC gathers job usage information and controls job limits, setting the PDC to run less frequently can result in information loss or overrun limits. If a job is short enough to start and complete between PDC runs, the PDC gathers no usage information for that job. In clusters that bill usage back to submitting users, this behavior may not be desirable. Whether or not to use this option depends heavily on the workload profile and how the cluster is managed.

Data Management

The Oracle Grid Engine software does not manage data. It is assumes the data required for a job is available on the execution host, and that the administrator has used the provided facilities to implement the required data staging. For clusters where some form of data staging has been implemented, the extensibility of the Oracle Grid Engine software's resource model presents some interesting opportunities for scheduling optimization, especially when dealing with very large data sets.

When jobs require large data sets, the time required to transfer the data to the execution host(s) can be a significant portion of the job's total run time. In cases where more than job or job run uses the same data, the time spent transferring the data is repeated needlessly. For data sets that are well-known, or can be constrained to a known directory structure, the administrator can create a custom resource to represent those data sets. Users can then submit jobs against those data set resources.

A boolean complex is sufficient for a well-known data set. It is recommended that a load sensor be configured on the execution host that reports the data set's presence. Jobs can be submitted with a soft resource request for that boolean complex. If an execution host is available that already has the data set, the job is scheduled on that system. Otherwise, the job is scheduled on a host without the data, and the data staging that was implemented for the cluster must stage the data onto that host. During the next load report interval, the load sensor for the boolean complex reports that this new host now also provides the data set.

For other data sets than can be confined to a known directory structure, a string complex can be used to model the presence of a number of data sets. It is recommended that a load sensor be configured on the execution host that reports the names of the root directories of the present data sets as a delimited string. (To simplify search strings, begin and end a delimited string with a delimiter and use a sorted order.) Jobs can request the string resource using wildcards to match the names of the data set(s) that the job needs.

Consider a complex named `datasets` and the "`-l datasets=*,mydata,*`" job request. If an execution host is available that already has the data set, the job is scheduled on that system. Otherwise, the job is scheduled on a host without the data, and the data staging implemented for the cluster stages the data onto that host. During the next load report interval, the load sensor for the string complex reports that this new host now also provides the new data set. To further simplify the user experience with this string complex solution, a job submission verifier (JSV) can be used to translate a more readable resource request, such as
"`-l datasets=mydata`" into the wildcard format.

High Availability

The Oracle Grid Engine software includes a built-in high availability mechanism called a shadow daemon or shadow master. The shadow daemon is a process that runs on a machine other than the master machine. As the `qmaster` process executes, it writes a monotonically increasing numerical value into a heartbeat file in the `qmaster` process spool directory. Every shadow daemon that runs in a cluster reads the heartbeat file periodically. If the file has not been updated since the last time it was read, a failover process is initiated.

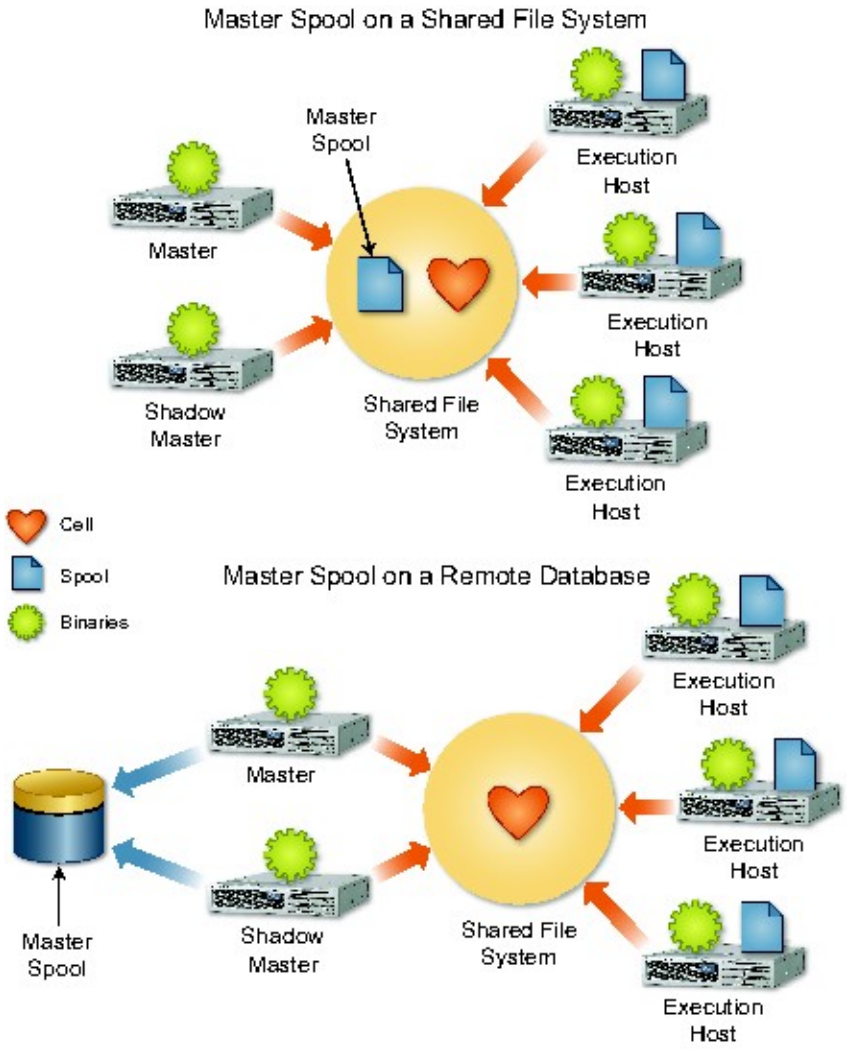

Figure 4-3. Placing the qmaster process spool directory on a shared file system or in a remote database can impact performance.

After several fail-safes to make sure the current `qmaster` process has stopped responding, one of the shadow daemon processes starts a `qmaster` process of its own on its local machine, making that system the new master. A file in the common directory contains the name of the current master machine. When a shadow daemon starts a new `qmaster` process, it updates the contents of that file. Whenever a client command or an execution daemon tries to connect to the `qmaster` process, it starts by looking in that file to determine which machine is the current master system. For this reason, a highly available cluster must place the `qmaster` process spool directory and the common directory on a shared file system.

In a large-scale cluster, employing high availability can introduce a significant performance penalty because of `qmaster` process data spooling. Neither a shared file system nor a remote Berkeley Database server offers the level of performance of a local data spool directory. One way to work around the data spooling bottleneck is to place the `qmaster` process and shadow daemon on systems

connected to the same dual-attached storage. Placing the `qmaster` process spool directory on a dual-attached storage unit means that both systems have access to the data, and both systems receive the performance benefits of local data access. In addition to increased hardware costs, this approach limits the cluster to a single shadow daemon. Under normal conditions, however, a single shadow daemon can suffice.

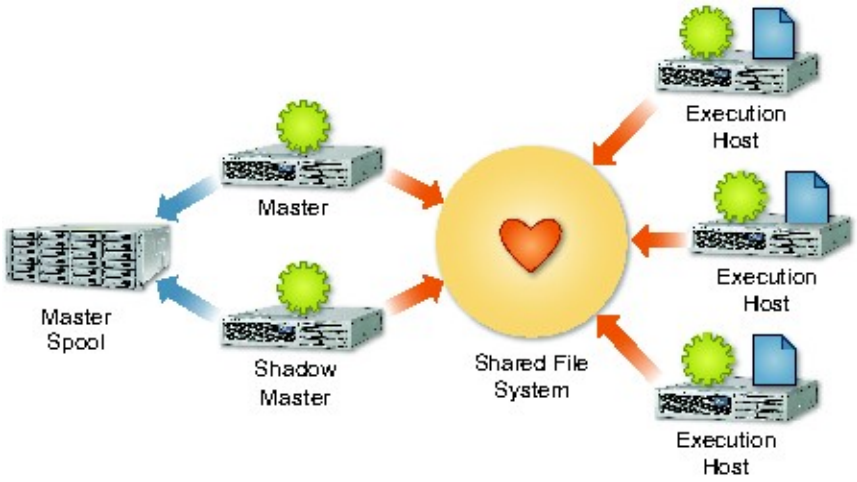

Figure 4-4. Placing the qmaster process spool on dual-attached storage can help improve performance.

Practices to Avoid

Several practices that are reasonable for smaller clusters should be avoided when building and running large-scale Oracle Grid Engine clusters.

- *Accounting files*

The `qacct` command is used to parse the accounting file to retrieve information about completed jobs. Every time a job finishes, a line is added to the accounting file with information about the job. In a cluster with very high job throughput, the accounting file can grow very quickly. For example, an accounting file with entries for 10 million jobs can cause the `qacct` command to take several minutes to run.

It is highly recommended that administrators rotate the accounting file periodically. However, rotating the accounting file results in job information being divided across several files. Rather than using the `qacct` command and the accounting file, use the Accounting and Reporting Console (ARCo) in large-scale clusters. ARCo stores job information in a relational database, making access to a large number of job data records reasonably fast. In addition, ARCo provides more information than the `qacct` command.

- *Cluster state*

The `qstat` command is used to monitor the state of the cluster. Unfortunately, processing a `qstat` request in a large-scale cluster can cause a noticeable burden on the `qmaster` process. If several users are running the command repeatedly to watch the progress of their jobs, the `qmaster` process can become unresponsive. Avoid encouraging users to use the `qstat` command. Instead, the -sync option to the `qsub` command often provides users with a reasonable alternative. The `-j` option to the `qstat` command can be used to gather information about a single job without burdening the `qmaster` process. Another option is to cache the output from the `qstat` command periodically and provide users a `qstat` wrapper that only reads data from the cache.

- *Scheduling information*

Prior to the Oracle Grid Engine software 6.2 release, the default configuration provided information via the `qstat` command about why pending jobs had not yet been scheduled. Starting in the Oracle Grid Engine software 6.2 release, that feature is turned off by default, as it can cause performance and scalability issues in larger clusters. While the feature can be turned back on, the practice is highly discouraged in a large-scale cluster. Instead, the Oracle Grid Engine software 6.2 update 2 release introduced a new option for the `qsub` and `qalter` commands (`-w p`) that allows a user to request scheduling information for a single job without impacting performance or scalability.

- *Job submission verification*

Job submission verifiers (JSVs) are a powerful tool introduced in the Oracle Grid Engine 6.2 update 2 release. Client-side and server-side JSVs are available. Client-side JSVs are executed by the client before a job submission is sent to the `qmaster` process. Server-side JSVs are executed by the `qmaster` process before accepting a job submission. Because client-side JSVs are implemented via a new command-line switch to the `qsub` command (`-jsv`), submitting users can replace or remove client-side JSV requests. Client-side requests can be removed using the `-clear` option.

It is tempting to use only server-side JSVs — but they carry a performance penalty. While care has been taken to make the server-side JSV framework as efficient as possible, an inefficient server-side JSV can have a severe performance impact on the `qmaster` process. Do as much JSV work as possible in client-side JSVs. It is recommended that server-side JSVs only perform verification that is critical to the cluster.

- *JSV creation*

Because the JSV framework communicates with JSVs through input and output streams, a JSV can be written in any language. The Oracle Grid Engine software includes C, Perl, Tcl, and Borne shell APIs to simplify JSV development by abstracting out the input/output protocol. Use of any shell scripting language, except possibly ksh93, is discouraged when writing server-side JSVs. Shell scripting languages lack many basic language primitives and rely on executing operating system commands. Every time an operating system command is executed, a new process must be launched. Depending on the platform, launching a new process can take several hundred milliseconds. If a JSV written in a shell scripting language executes just a few commands, it can add more than a second to the time to accept every job submission. In a cluster with high job throughput, such a server-side JSV script easily can reduce the cluster's job throughput rate to less than one per second.

Conclusion

The Oracle Grid Engine software is a highly scalable and flexible system for managing workloads on a wide variety of clusters, ranging from tiny single- or dual-node clusters to huge super-scale clusters like the TACC Ranger cluster with almost 4,000 hosts and over 63,000 processor cores. While the Oracle Grid Engine software is capable of scaling to meet the largest super-computing challenges, some care must be taken in cluster construction and configuration to unlock the software's full potential. The tips and suggestions presented in this document are aimed at helping readers approach the building of a super-scale or large-scale compute cluster.

About the Author

Daniel Templeton Product Manager for the Oracle Grid Engine product. Formerly a developer on the Oracle Grid Engine and Service Domain Manager products, Daniel's main contributions were to the Oracle Grid Engine DRMAA implementations and the architecture and design of the Service Domain Manager product. In his current role, he acts as both a customer and community advocate, working to drive interest, adoption, and customer satisfaction for the Oracle Grid Engine and Service Domain Manager products.

Acknowledgments

The author would like to thank Roland Dittel, Andreas Haas, Michael Schulman, and Andy Schwierskott for their feedback and contributions to this paper.

Appendix A: Glossary

BATCH JOB

A single segment of work. See *job*.

CELL DIRECTORY

A directory under the root installation directory that contains all of the information about a cluster.

CLUSTER

A group of hosts connected together by the Oracle Grid Engine software.

COMMON DIRECTORY

A directory in the cell directory that contains a variety of data and scripts that are useful to many components in a cluster.

COMPLEX

A resource modeled in a cluster.

COMPUTE CLUSTER

A cluster that is focused on providing compute resources. See *cluster*.

CONSUMABLE

A complex whose value is decremented when jobs that request it are scheduled. See *complex*.

EXECUTION DAEMON

The component responsible for executing jobs using compute resources.

EXECUTION HOST

A host on which an execution daemon is running.

INTERACTIVE JOB.

Cluster-managed interactive login. See *Job*.

JOB

A segment of work to be executed by the cluster, defined as anything executable from a command terminal.

JOB SLOTS

The capacity of a queue to run a job. The number of job slots available on a machine often correlates to the number of processor cores.

LOAD SENSOR

An administrator-defined executable that measures and reports the value of one or more complexes. See *complex*.

MASTER HOST

The host on which the qmaster process is running.

PARALLEL JOB

A job composed of cooperating distributed tasks. See *job*.

PARAMETRIC OR ARRAY JOB

A group of similar work segments operating on different data. See *job*.

QMASTER

The central component of a cluster.

RESOURCE

See *complex*.

SCHEDULING INTERVAL

The period between successive scheduling runs. See *scheduling run*.

SCHEDULING RUN

A single execution of the scheduler in which the entire list of pending jobs is resorted according to policies, and available job slots are filled with jobs from the top of the sorted list.

SHADOW DAEMON

A daemon that manages `qmaster` process failover.

SHADOW MASTER

See *shadow daemon*.

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