Comparative Study of Distributed Resource Management Systems – SGE, LSF, PBS Pro, and LoadLeveler

Yonghong Yan, Barbara Chapman
{yanyh,chapman}@cs.uh.edu

Department of Computer Science
University of Houston

Abstract

Distributed Resource Management Systems (D-RMS) control the usage of hard resources, such as CPU cycles, memory, disk space and network bandwidth, in high-performance parallel computing systems. Users request resources by submitting jobs, which could be sequential or parallel. The goal of a D-RMS is to achieve the best utilization of resources and to maximize system throughput by orchestrating the process of assigning the hard resources to users' jobs. In the past decade, lots of work have been done to survey and study those systems, but most of them are from the viewpoint of users and D-RMS provided functionalities. In this study, we comparatively study current widely-deployed D-RMS from the system aspect by decomposing D-RMS into three subsystems: Job management subsystem, physical resource management subsystem, and scheduling and queuing subsystem. Also the system architecture to organize these three subsystems are discussed in detail. This work contributes to the D-RMS vendor and research in distributed resource management by presenting D-RMS internals for the designer in their future system upgrade and improvement.

1 Introduction

Distributed Resource Management Systems (D-RMS) control the usage of hard resources, such as CPU cycles, memory, disk space and network bandwidth, in high-performance parallel computing systems. Users request resources by submitting jobs, which could be sequential or parallel. The goal of a D-RMS is to achieve the best utilization of resources and to maximize system throughput by orchestrating the process of assigning the hard resources to users' jobs. In the literature, D-RMS are also called job management systems, resource management systems, schedulers, queue systems, or batch systems. In this study, we use Distributed Resource Management Systems (D-RMS) to represent them. Some widely deployed D-RMSs include Platform Load Sharing Facility (LSF) [16], Portable Batch Systems (PBS) [2, 1], Sun Grid Engine (SGE) [19], IBM Load Lever [10], and Condor [20].

Two reports from NASA [12, 17] enumerate the requirements of a D-RMS to support NASA parallel systems and clusters. A very detailed requirement checklist to help evaluate D-RMS is presented in [12]. Two Phase I evaluations of selected RMS softwares were performed against these requirements [11, 13]. The selected D-RMS are CODINE (SGE), DQS, LoadLeveler, LSF, NQE and PBS. Other reports in the mid 90s [14, 3] performed a piece-by-piece comparative study of features of a rather comprehensive list of D-RMS. Some studies for special needs or features have also been done. For example, to support the Sun HPC ClusterTools parallel execution environment [15], Hassaine [8] analyzed issues in selecting a D-RMS in clusters. A tutorial study about integrating Sun HPC ClusterTools in LSF, GRD and PBS was performed in [4]. Steve [18] presented an integration architecture for Sun HPC ClusterTools and cluster D-RMS. Recently, a detailed comparative study and ranking were performed on the popular D-RMS [5, 7]. A performance study of LSF, PBS, SGE and Condor has been reported in terms of system throughput, average turn-around time, average response time and resource utilization [6].

These excellent studies summarized the required feature list in D-RMS and some of them ranked those systems based on their requirement, which is very helpful for guiding site administration in choosing the right D-RMS for their systems. Other studies reported the integration and usage experience of D-RMS to meet site-specific needs. But these studies address little regarding D-RMS internal and this is desirable so that the D-RMS systems can be viewed from a high-level and implementation point of view in addressing challenges in resource management research for high performance large-scale systems.

Till now, development of D-RMS are mostly from industry vendor with focus on feature additions based on current systems that were designed before to manage moderate-sized systems. And for the next-level of scale in high-performance systems, either petascale or extreme-scale in some context, we found few research projects addressing resource management issues. Some of them, which are very new, are Scalable Systems Software (SSS) project of SciDAC and HPC Colony of DoE OS/Run-Time program. Started on 2001, SciDAC SSS addresses the scalability issues of resource management by decomposing the RM into functional components interfacing using XML standard. SSS relies on an MPD-patched PBS for job control. The best information we can get for the HPC Colony project, which was announced on SC04, is to create services and interfaces for parallel resource management and global system management to support high-performance computing systems with very large numbers of processors [9].

This comparative study work goes in depth into the system architecture and related issues, modularization, and advanced features in current D-RMS in the following aspects:

- Each of the studied systems (SGE, LSF, PBS Pro, and LoadLeveler) are presented in the proper detail.
- Modularize D-RMS in three subsystems: Job Management subsystems, Physical Resource Management subsystem, and Scheduling and Queuing subsystems. Feature lists of the three subsystems are presented by combining the provided functionalities from each D-RMS.
- Scheduling and queuing are discussed and compared in detail in the studied systems.

We believe that this study will contribute for both the D-RMS vendors to improve their systems, and also for the D-RMS research for future large-scale and petascale systems. The paper is organized as follows. In next section, we will have an introduction of D-RMS, including a modularized view of D-RMS and the details of each module. Then in another four section, SGE, LSF, PBS Pro and LoadLeveler are studied in detail. We conclude our work in section 7.

2 Distributed Resource Management System

Distributed Resource Management Systems (D-RMS) assigns computational resources to resource users within one administrative domain, the most-common and typical example of which is a cluster. As mentioned before, the goal of a resource management system is to achieve the best utilization of resources and to maximize system throughput by effectively managing hard resources and users' jobs and by assigning resources to jobs. This assignment, also referred to as scheduling, makes decisions about where to launch users' jobs and how many hard resources are allocated to them. In this sense, a D-RMS can be modularized into three subsystems: job management subsystem, physical resource management subsystem, and scheduler and queue subsystem.

2.1 Job Management Subsystem

Job Management subsystem (JMS) is the interface of resources for the users who interact with RMS by requesting resources via job submission, control and monitor their jobs, etc. JMS also takes care of the job launching, process or thread control of the launched jobs. In supporting this, JMS should be able to manage different types of jobs, like simple script jobs, interactive jobs, parallel jobs developed using MPI, OpenMP, or other types of parallel libraries, job arrays and complex dependent jobs. The functionalities provided by JMS should includes:

- Job (Re)Submission: Options when submit or alter a job can be categorized as following:
  - Options for job specification, including job executables, job type, job name, job arguments, input/output file, ...
  - Options for job execution environment setup, such as queues, parallel environment, notification events, priority, ...
  - Options for job resource requirements and limits, including number of CPUs, wall-clock time, CPU time, memory, swap and disk space, network bandwidth, and resource access type(dedicated or shared).
  - Options for scheduling, security and others
- Job Alter, Rerun, and Re-queue, event notification
- Job Status Checking. A job could be in state as Submitted, Hold, Run, Done, Suspended, Exit, etc
- Job Holding/Suspension/Resuming
- Job Signaling
- Job (Re)Prioritizing
- Job Accounting: Recording the resource usage by users or jobs.
- Job checkpointing, restarting and migrating for fault-tolerance and load-balance purposes
- Job Performance Fault handling
- Customized Job Starter, Job's Pre-Execution and Post-Execution, and other customized job control actions, especially for parallel jobs, customized application execution environments.

JMS should support various types of jobs, such as batch/script jobs, interactive (X-Windows) jobs, array jobs, parallel jobs in MPI, PVM, OpenMP, and other multi-thread model, and dependent jobs. User interfaces should be kept same for the user but JMS also recognize the differences of them. Also, JMS should support for job's file staging-in and staging-out. File "stage-in" and "stage-out" capability allows the user to identify files that should be transferred to and from appropriate locations on the computer system on which his/her job will be run. Stage-in needs to occur before the actual job starts, but after disk resources have been allocated for that job. Stage-out should follow termination of the job, but before the disk resources are released for re-allocation. [5]

2.2 Physical Resource Management

There are two major tasks in physical resource management, first, to control the usage of the hardware resources, such as CPU cycles, memory, swap area, disk space, and network bandwidth, and second, to report and account the status and usage of the resources. The resource control applies the resource usage constrains or local usage policy, and mostly called by the job execution daemons or utilities, or the event handler, which is triggered by resource monitoring systems. The status and account information of the resources are used for scheduling purpose and status checking by the users or system administrators. Concerning about the resource information, they are summarized below:

- Static Information
  - Cluster architecture, such number of nodes, number of CPUs per nodes, node and CPU detail(architecture, OS, etc)
  - Memory architecture, shared memory space, available space, etc
  - Network, network topology, bandwidth, latency
  - Software resources, such as shared library and utilities, compiler, JVM, Grid resources, etc
- Dynamic Information
  - Resource Load information and thresholding, normalized load value for comparison
  - Memory usage percentage
  - Network bandwidth available
  - Normalized general resource usage value

2.3 Scheduler and Queue Systems

In RMS, resource assignment must respect site resource usage policies, which are carefully handled by different complicated scheduling algorithms, must recognize administrative domain boundary by enforcing security when a scheduling has to across domains (this is must related to grid-level scheduler). In high-end computing and for best performance boasting of hard resources, the scheduler and queue system must recognize the performance enhancement of special hardware and network, such as multi-NIC on nodes, high-performance networking, network topology, multi-core node, cache and memory hierarchy, and parallel file systems. Those features of hardware performance enhancement must be taken into consideration in scheduling process for best system throughput and application performance.

Queue subsystem organizes the holding jobs in different groups. A well-designed queue system can greatly reduce the average job waiting time and the average idle time of system resources. Queues are created to best utilize hard resources, and match various types of jobs with jobs' favorite resources. For example, a queue for the nodes the cluster that connect to the same network switch, and a parallel applications executed on this queue will enjoy the high-speed intra-switch communication; or a queue consisting of nodes with higher speed network to file server and some I/O-intensive jobs will gain good performance if scheduled on it; or a queue created specifically for short jobs, or interactive jobs that need rapid turn-around time; or a queue created with space-shared scheduling policy for the jobs that want exclusively access on the resources; or a queue created with time-shared scheduling policy for the jobs that can share the resources during executions; or some temporary queues are created specifically for resource reservation purpose.

In the rest of this report, some widely-deployed D-RMS: SGE, LSF, PBS Pro, and LoadLeveler will be discussed.

3 Sun Grid Engine (SGE)

Sun open-source Grid Engine is a product migrated from Distributed Networked Environment (CODINE)/Global Resource Director(GRD), which originated from Distributed Queue Systems(DQS). SGE is Sun's major strategical step toward grid computing by deploying SGE onto a campus grid.

3.1 Software and Installation architecture

SGE daemon installation architecture is shown in Figure 1. The master host with sge_qmaster and sge_schedd daemons is the control and database server of a SGE administrative unit, which is called "cell" by SGE. Sge_qmaster daemon holds the most current data about states of jobs, queues, resources and other necessary objects, and also answers all client requests for job control and resource control. The scheduler daemon (sge_schedd) maintains job queues and assigns suitable resources (hosts, queues) to jobs waiting for execution. SGE-supported scheduling policies includes FCFS, priority-based like sharetree, functional, deadline and override in SGE terms. We will get into detail of them in later section. The execution hosts, which are the SGE-managed computational resources have sge_execd daemon installed for both the management of jobs and hard resources. Sge_execds report information about resource load, job execution to SGE master daemon, and also launch jobs forwarded from sge_qmaster. To prevent from qmaster becoming a single point of failure, A shadow master host running sge_shadowd daemon monitors the master host and starts a new qmaster-schedd pair if it detects failure of the daemons or host. A typical SGE installation has system configurations and database spooling file on the file server accessible from SGE hosts.

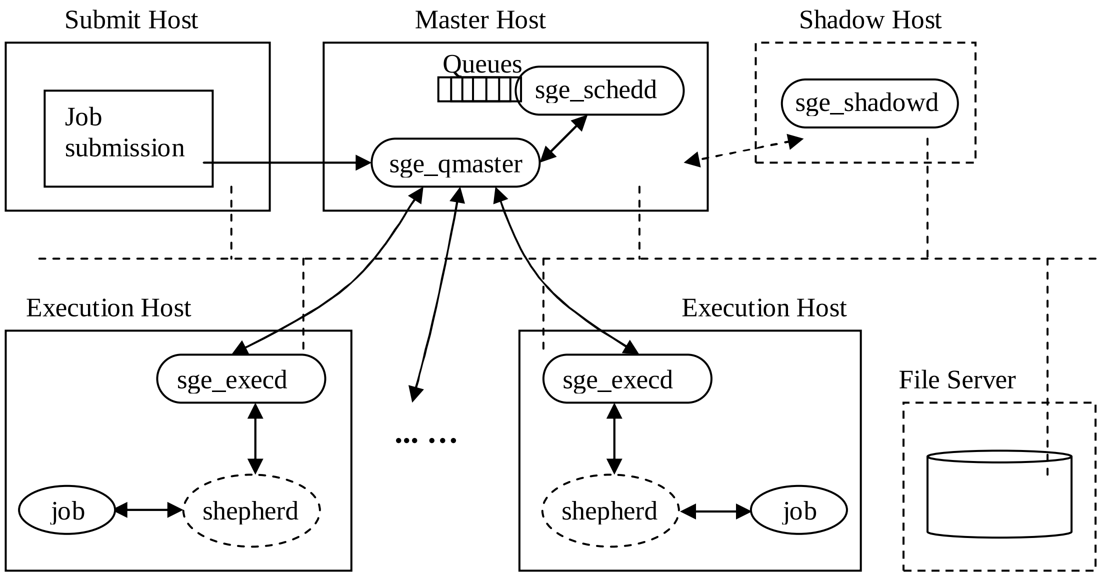

Figure 1: Sun Grid Engine Installation Architecture

While sge_qmaster answers job submission and query request from users, sge_execd is responsible for launching jobs and control of it. Also instead of fork the job's process directly, sge_execd creates a sge_shepherd daemon for each job runtimely. After setting up the job execution environment, sge_shepherd starts the job, and then performs job control tasks forwarded from sge_execd during the job's execution. When job stops, sge_shepherd collects job exit status, reports it to sge_execd and cleans up the job environments. Sge_execd also performs tasks related to physical resource control, accounting and monitoring. To report runtime load value, one function called load_sensor in sge_execd retrieves the status and actual load values of the execution host and send them back to the sge_execd. The sge_execd reports the information to sge_qmaster for scheduling purpose.

3.2 Scheduling and Queue Management

Sge_schedd makes decisions about resource allocation in SGE. After registering itself with qmaster and before each starting scheduling run, schedd updates its internal data pushed from qmaster. These data are structured as an event list consisting of resource status, holding jobs and queues, etc. In each scheduling run, schedd generates a so-called order for each decision about job resource assignment, and then sends a list of orders to qmaster which will take actions upon these orders.

SGE scheduler was designed to be easily customized in terms of scheduling algorithms within current SGE feature set. But to extend current features, including associated algorithms is not possible in many cases without changing or enhancing the data structures with new parameters [19]. The main reasons are mentioned in the internal documentation of sge_schedd: "This implies the need for administering these new parameters and usually makes modification in the user interfaces necessary. To store the settings of such new parameters to disc, it becomes necessary to change the file format in which qmaster and other components store their state. And changing the file format makes it necessary to think about how to migrate from an existing Grid Engine installation." Further, extending or adding new scheduling algorithms have to be done in source code level instead of API. This could be considered a major drawback in terms of scheduler extensibility, and lessons should be learned in future design of an extensible systems. That is how to separate feature extension with internal data structure and persistence storage of data.

But current SGE default scheduler is a powerful one that is enough for most of the administration sites. Two policy modes are support in current Grid Engine, SGE mode and SGEEE mode. In SGE mode, either FCFS policy or user_sort policy which tries to be fair in terms of number of jobs of each users can be configured, and priority setting can override FCFS and user_sort.

In SGEEE mode, a job's priority is set according to four high-level policies: share tree, functional, deadline, and override. In share-tree policy, the entitlements of resource usage to different project groups and users is organized in a hierarchical tree and leaves of the tree are individual projects and/or users. Each subtree (including leaves) is entitled to receive some percentages of all the resources of what its root receives from higher level. Functional entitlements represent a kind of relative priority and are defined based on functional attributes like being a certain user or being associated with a project or a certain user group. In deadline policy, the entitlements of deadline jobs grows automatically as they approach their deadline. The above three automatic policies can be overridden manually via the override policy, which allows, for instance, to double the entitlement which a job (but also an entire project or one user) currently receives. It is also often used to assign a constant level of entitlement to a user, a user group or a project. For share-tree and functional, different weights can also be given to favor one of them. Combining the four policies, tickets are assigned to jobs and jobs with highest amount of tickets are investigated by the scheduler first. So it is also a priority-based policy, but the priority is dynamically changed.

Schedd fills queues with jobs in the order of either sequence number or the load of the queue. Sequence number of a queue is given by system admin. The load of the queue is represented by a simple algebraic expression used to derive a single weighted value from all or part of the load parameters of a queue reported by sge_execd for each host and from all or part of the consumable resources being maintained for each host. So the load of a queue is highly configurable by means of a flexible algebraic expression on the queue parameters. Also, calendar is used to specify "on duty" and "off duty" time periods for queues on a time of day, day of week or day of year basis. By setting "off duty" in the queue calendar on the routine system-maintenance periods can stop queuing of incoming jobs.

3.3 Parallel Applications Support

To support various types of parallel applications, SGE provides an object called parallel environment (PE) which must be used when submitting a parallel jobs. SGE PE captures the information about amounts of computation slots, user/group access control, customized PE startup/shutdown procedure and related parameters, and slot allocation methods such as fill_in, round_robin. SGE PE helps the scheduler on the resource allocation for parallel jobs. Concerning about job control (basically job startup and stop), SGE integrates with client commands of implementation of MPI, PVM and multi-thread models. The integrations are script-based and loosely coupled with the help of SGE PEs. In these methods, the major task is to extract a list of hosts from the PEs that job is submitted to and scheduled on . From this host list, job launching commands (such as mpirun) will be issued. Scripts for other job control (such as suspend and resume) are also provided (specially for Sun MPI) and these scripts wrap MPI-provided commands and are configured in the related fields of queues. When the user perform these job control, the customized scripts will be called.

Sun also developed a tight-integration between D-RMS with Sun Cluster Runtime Environment (CRE) [18], which supports for launching, stop, and monitoring of parallel jobs, specifically for Sun MPI.

The disadvantages of loosely coupled integration are two folds, first, the scripting varies with different parallel programming models and there is no uniform interfaces for scripts; second, the spawned processes escapes the control and monitoring of SGE after launching a job.

4 Load Sharing Facility (LSF)

LSF from Platform Computing is a mature product set including LSF, LSF HPC and other complementary and additional products. In our study, we focus on LSF HPC that is most comparable in our studies.

4.1 Software and Installation architecture

LSF is the most complicated surveyed systems regarding the daemon installation and setup as shown in Figure 2. On each server host which could be a master host, an execution host or a submission host, at least four LSF daemons are required to installed: sbatchd, lim, pim, and res. Sbatchd (Slave Batch Daemon) and res (Remote Execution Server) perform tasks related to job execution management; Sbatchd is also responsible for enforcing local policies; lim (Load Information Manager) collects host load and configuration information and reports them to master LIM running on the master host; pim (Process Information Manager) started by lim collects resource usage information about the job processes and reports them to sbatchd. On the master host, mbatchd (Master Batch Daemon), mbschd (Master Batch Scheduler Daemon) and Master LIM are installed to control the overall management and scheduling activities of the whole system. Started by local sbatched, mbatchd answers requests from users, manages jobs held in queues, and dispatches jobs to hosts as determined by mbschd, which makes scheduling decisions based on job requirements, resource load status and policies. Master LIM receives load information from LIMs running on each server host and forwards the information to mbschd via mbatchd to support scheduling decisions. Other than LSF LIM on each server host, an External LIM (ELIM) can be installed to collect and track customized dynamic load information. ELIM is a site-definable executable that could be a script or binary program.

4.2 Scheduling and Queuing

In LSF, there are two stages in the decision-making process of resource assignments for the submitted jobs: queuing and scheduling. When submitted, jobs are put and held in a queue specified by users or the system default one. The queuing step impacts the jobs' dispatching time because LSF tries to schedule jobs from the highest priority queue first. Queues implement different job

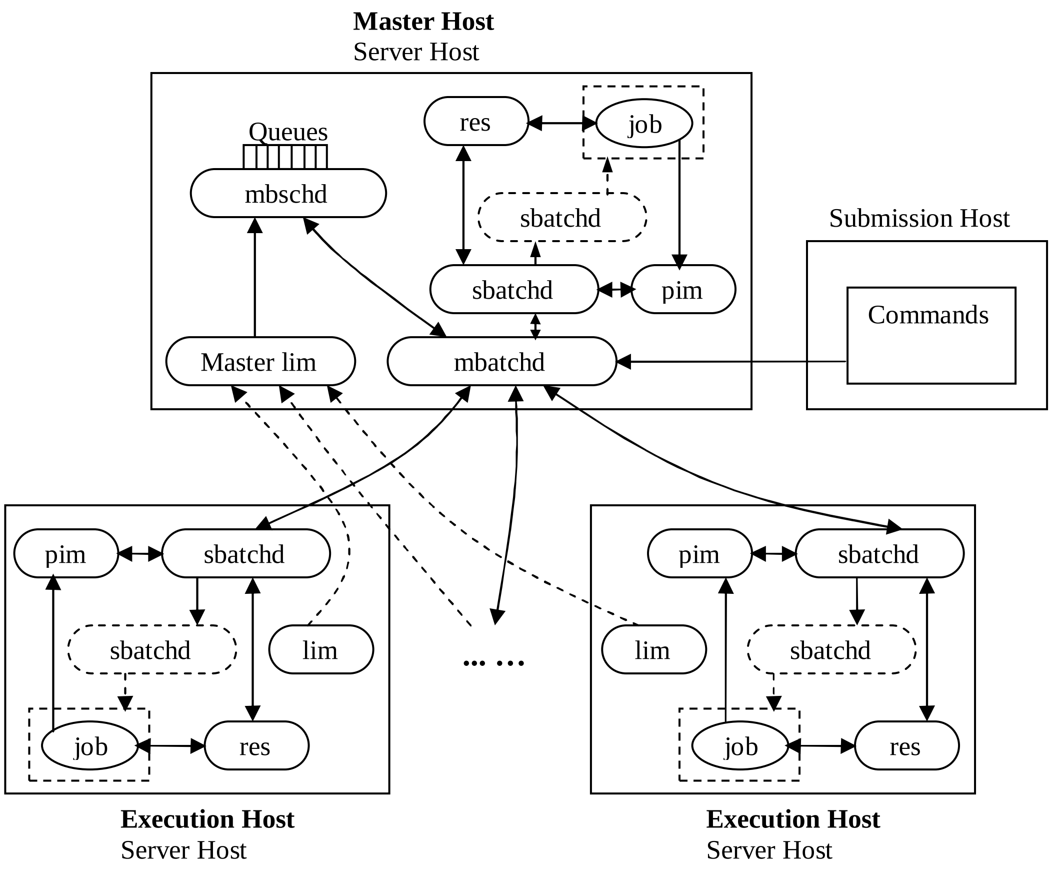

Figure 2: LSF Installation Architecture

scheduling and control policies. All jobs submitted to the same queue share the same scheduling and control policy. Queues do not correspond to individual hosts; each queue can use all server hosts, or a configured subset of the server hosts. Queues have the following attributes associated with them:

- Priority, where a larger integer is a higher priority
- Name, which uniquely identifies the queue
- Queue limits, that restrict hosts, number of jobs, users, groups, processors, etc.
- Standard UNIX limits: memory, swap, process, CPU, etc.
- Scheduling policies: FCFS, fairshare, preemptive, exclusive
- Administrators
- Run conditions
- Load-sharing threshold conditions, which apply load sharing to the queue

- UNIX nice value, which sets the UNIX scheduler priority

There are several scheduling policies in LSF: the simple FCFS, Fairshare scheduling, Deadline and exclusive scheduling, preemptive scheduling, and SLA-driven scheduling.

The deadline constraint scheduling in LSF is different from SGE deadline. LSF deadline is for queue and if a job cannot complete before the queue deadline, it will be suspended. For SGE, the deadline is for job and if a job is approaching its deadline, the system will increase its priority so that it can be scheduled to run earlier. Exclusive scheduling allows a job to use the hosts exclusively during its execution. LSF dispatches the job to a host that has no other jobs running, and does not place any more jobs on the host until the exclusive job is finished. Preemptive scheduling lets a pending high-priority job take resources away from a running job of lower priority. When two jobs compete for the same resource, LSF automatically suspends the low-priority job to make resources available to the high-priority job. The low-priority job is resumed as soon as possible.

Fairshare scheduling divides the processing power of a cluster among users and groups to provide fair access to resources. Fairshare can be defined at either the queue level or the host level. However, they are mutually exclusive and queue-level fairshare and host partition fairshare cannot be configured in the same cluster. If you want a users priority in one queue to depend on their activity in another queue, you must use cross-queue fairshare or host-level fairshare. Each fairshare policy assigns a fixed number of shares to each user or group. These shares represent a fraction of the resources that are available in the cluster. The most important users or groups are the ones with the most shares. Users who have no shares cannot run jobs in the queue or host partition. A users dynamic priority depends on their share assignment, the dynamic priority formula, and the resources their jobs have already consumed. The order of jobs in the queue is secondary. The most important thing is the dynamic priority of the user who submitted the job. When fairshare scheduling is used, LSF tries to place the first job in the queue that belongs to the user with the highest dynamic priority. Concerning this, LSF fairshare is similar to SGE share-tree policy.

Fairshare policy configured at the host level handles resource contention across multiple queues. You can define a different fairshare policy for every host partition. If multiple queues use the host partition, a user has the same priority across multiple queues. Fairshare policy configured at the queue level handles resource contention among users in the same queue. You can define a different fairshare policy for every queue, even if they share the same hosts. A users priority is calculated separately for each queue. Fairshare policy configured at the queue level handles resource contention across multiple queues. You can define a fairshare policy that applies to several queues at the same time. You define the fairshare policy in a master queue and list slave queues to which the same fairshare policy applies; slave queues inherit the same fairshare policy as your master queue. For job scheduling purposes, this is equivalent to having one queue with one fairshare tree. In this way, if a user submits jobs to different queues, user priority is calculated by taking into account all the jobs the user has submitted across the defined queues. In hierarchical fairshare, for both queue and host partitions, if groups have subgroups, additional levels of share assignments can be configured , resulting in a multi-level share tree that becomes part of the fairshare policy.

In priority-based queue system, a high priority queue tries to dispatch as many jobs as it can before allowing lower priority queues to dispatch any job. Lower priority queues are blocked until the higher priority queue cannot dispatch any more jobs. However, it may be desirable to give some preference to lower priority queues and regulate the flow of jobs from the queue. Queue-based fairshare allows flexible slot allocation per queue as an alternative to absolute queue priorities by enforcing a soft job slot limit on a queue. This allows you to organize the priorities of your work and tune the number of jobs dispatched from a queue so that no single queue monopolizes cluster resources, leaving other queues waiting to dispatch jobs.

You can balance the distribution of job slots among queues by configuring a ratio of jobs waiting to be dispatched from each queue. LSF then attempts to dispatch a certain percentage of jobs from each queue, and does not attempt to drain the highest priority queue entirely first. When queues compete, the allocated slots per queue is kept within the limits of the configured share. If only one queue in the pool has jobs, that queue can use all the available resources and can span its usage across all hosts it could potentially run jobs on.

5 Portable Batch Systems Professional Edition (PBS Pro)

PBS Pro is the enhanced commercial version of PBS, originally developed from NASA to meet the needs of distributed parallel support which was lack in the first generation queuing batch systems, the Network Queuing Systems (NQS).

5.1 Software and Installation architecture

Compared with LSF, PBS architecture keeps the minimum necessary daemons in the installation, as shown in Figure 3. On the master host, which is called server host in PBS, only pbs_server and pbs_sched are installed, responsible for answering user request and scheduling. On the execution host, pbs_mom is the only daemon taking care of job launching, enforcement of site policies and managing physical resources.

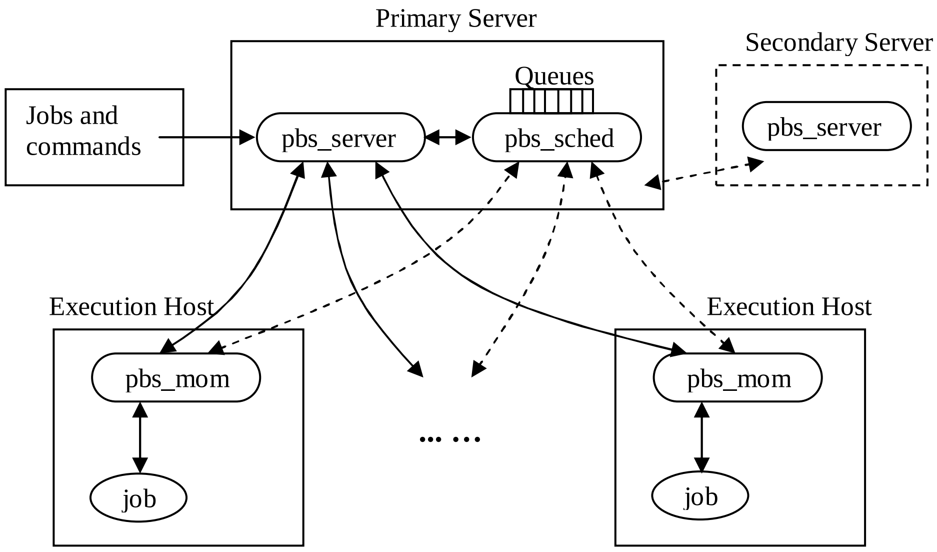

Figure 3: PBS Pro Installation Architecture

5.2 Scheduling and queuing

PBS Pro supports all the general policies in scheduling, such as FIFO, user or group priority-based, faireshare, preemptive and backfilling. By default, jobs in the highest priority queues will be considered for execution before jobs from the next highest priority queue; but it is configurable as a round-robin fashion that queues will be cycled through attempting to run one job from each queue. There are two types of queues, routing and execution. A routing queue is used to move jobs to other queues including those that exist on the different PBS servers. A job must reside in an execution queue to be eligible to run and should remain in an execution queue during the time it is running.

6 LoadLeveler

LoadLeveler was originally developed to exploit spare cycles on workstations, which are typically idle for much of the time when not actively being used. Now LoadLeveler is a customized resource management systems for IBM SMP and clusters and can be installed on IBM AIX, and also Linux with some functionality restriction. LoadLeveler integrates with lots of low-level modules of AIX, such as AIX Workload Manager (WLM), AIX checkpointing/restarting tools to provide a finer-grained resource control over IBM systems.

6.1 Software and Installation architecture

The architecture of LoadLeveler has little difference and is relatively more scalable than other systems, see Figure 4. The LoadLeveler-called master daemon (LoadL_master) running on each execution host manages the daemons on it. It is not the real "master" in other D-RMS that performs the centralized management tasks and answers users' requests. But on the host designated as central manager, the master runs the negotiator daemon which is the true "master" holding the overall system data and deciding the global scheduling assignment. Another daemon that may be confused with the corresponding one in other D-RMSs is the scheduler daemon (LoadL_schedd). LoadL_schedd can be configured to run on any execution host. Instead of performing tasks of resource allocation, LoadL_schedd answers job submission requests from users and forwards them to the negotiator (LoadL_negotiator) on the central manager host, which does the real decision-making about resource allocation. The negotiator also collects resource information used for scheduling, keep system-wide data objects and performs other system-wide tasks and coordination. LoadL_schedd can be configured on multiple execution hosts to answers users' requests and queues jobs that are submitted to it. This mechanisms are very helpful to the negotiators a lots and are very scalable by distributing the system wide tasks of scheduling and answering request to different daemons.

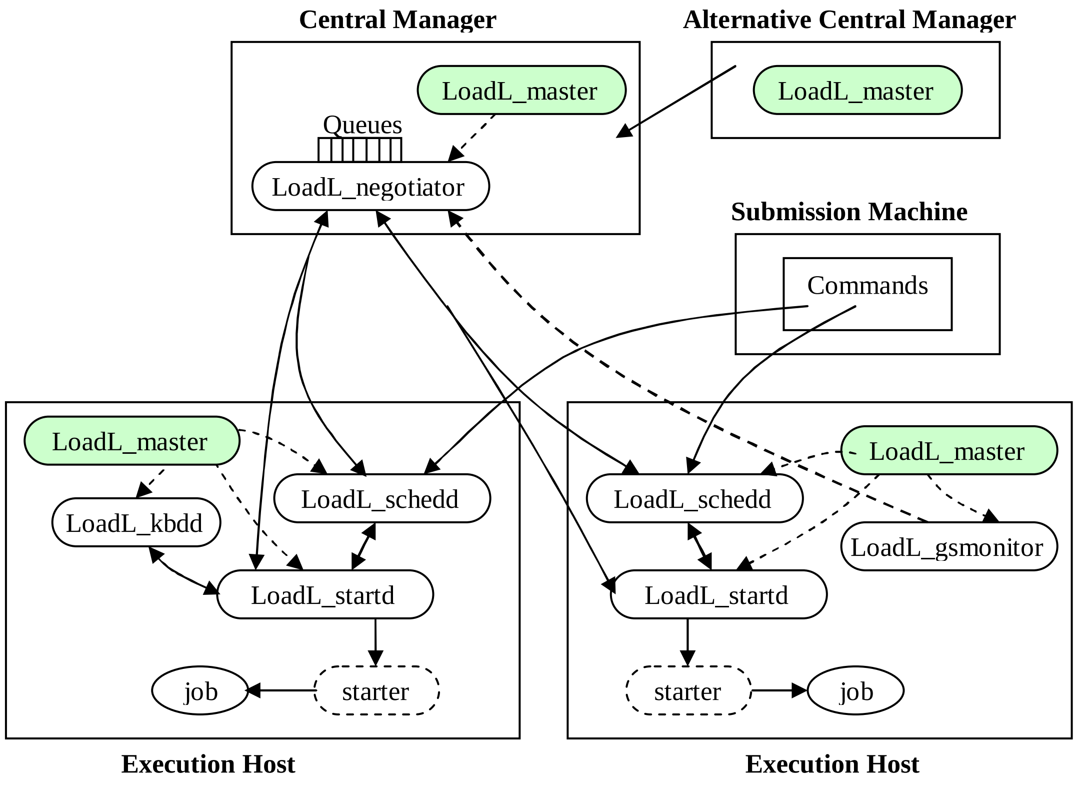

Figure 4: LoadLeveler Installation Architecture

6.2 LoadLeveler Scheduler and Queue Subsystems

LoadLeveler use term "class", instead of queues for user to submit jobs. A job class is use to manage different types of jobs. For example, a class created for short running jobs or jobs requiring quick turn-around time. The system administrator can define these job classes and select the users that are authorized to submit jobs of these classes. Queuing is implemented internally, A job queue is a list of jobs that are waiting to be processed. When a user submits a job to LoadLeveler, the job is entered into an internal database which resides on one of the machines in the LoadLeveler cluster until it is ready to be dispatched to run on another machine.

LoadLeveler supports three internal schedulers: the default scheduler (LL_DEFAULT), BACKFILL scheduler, and GANG scheduler. Also, LoadLeveler provides a functional API so that external scheduler can be tapped into. Currently, Maui scheduler is integrated with it.

The default scheduler: LL_DEFAULT, has been available since the produce was first introduced. For serial jobs, LoadLeveler simply looks at the jobs at the top of the queue, and if there is a node available that satisfies any additional requirements, then the job will run. For parallel jobs, the default scheduler attempts to accumulate sufficient resources for it by reserving the available and newly released resources. The total amount of time that the job is allowed to reserve nodes is a configurable parameter to make sure that idle resources are not held too long for jobs with many resource requests.

There are two major disadvantages with this scheme. Firstly, as more and more nodes are reserved for a parallel job, the system as a whole becomes progressively less utilized. And secondly, parallel jobs often never manage to accumulate enough nodes within the allowed holding time [10].

The BACKFILL scheduler provides means to fill the idle cycles of the resources reserved by parallel jobs and thus eliminates the disadvantages of the default scheduler. For parallel jobs, by looking ahead and determining (if possible) the time when the required resources will be available, the BACKFILL scheduler calculates how long of the current available resources will be held and idle before used by the jobs. The scheduler then looks for smaller jobs who promise to be finished within the calculated idle period, and dispatch them to fill those idle resources. To help this prediction and be backfilled, jobs should provide the wall clock limit of its execution, which is one of the main differences between BACKFILL and the LL_DEFAULT scheduler from the users point of view.

GANG Scheduler Gang scheduling allows two or more jobs to be simultaneously active on a node and for execution to swap between the jobs at regular intervals. In the case of a parallel jobs, the switching between tasks is coordinated across the cluster. The number of columns is determined by the number of processors under the control of the GANG scheduler.

The GANG scheduling [21] was first introduced in production system like LoadLeveler. The GANG Scheduler uses a two-dimensional GANG matrix where columns represent processors, and rows represent time slices. The interval of the time slice is set by the LoadLeveler administrator to be the minimum length of execution for jobs such that the overhead of context switch does not significantly impact the overall system performance.

7 Conclusion

This study work provides a detail analysis of the internal architectures in D-RMSs by decomposing a D-RMS into three modules: Job management subsystem, Physical resource management system, and scheduling and queuing subsystem. These three modules, together with the architecture in organizing them in each of our surveyed system are studied in detail. Since these systems are rather complex, to present them in a relatively detail and compare them require much more work than what we have done and this will be our future direction.

Acknowledgments

We greatly appreciate Dr. Cummings of her guidance on the organization and expression of the papers. Without her help, it is impossible for us to present this work so thoroughly and clearly.

References

[1] Altair Grid Technologies. OpenPBS: Open Portable Batch System. `http://www.openpbs.org`.

[2] Altair Grid Technologies. PBS Pro: Portable Batch System Professional Edition. `http://www.pbspro.com`.

[3] Mark Baker, Geoffrey Fox, and Hon Yau. Cluster Computing Review. Technical Report SCCS-748, Northeast Parallel Architecture Center, Syracuse University, USA, November 1995.

[4] Chansup Byun, Christopher Duncan, and Stephanie Burks. A Comparison of Job Management Systems in Supporting HPC ClusterTools. In *Technical Forum of Sun Users Performance Group-SUPerG*, Fall 2000. `http://www.sun.com/datacenter/superp`.

[5] Tarek El-Ghazawi, Kris Gaj, Nikitas Alexandridis, and Brian Schott. Conceptual Comparative Study of Job Management Systems. Technical report, George Mason University, Feburary 2001. `http://ece.gmu.edu/lucite/reports.html`.

[6] Tarek El-Ghazawi, Kris Gaj, Nikitas Alexandridis, Frederic Vroman, Nguyen Nguyen, Jacek R. Radzikowski, Preeyapong Samipagdi, and Suboh A. Suboh. A Performance Study of Job Management System. *CPE Journal*, 2003. `http://ece.gmu.edu/lucite/publications/CPE_journal_2003.pdf`.

[7] Kris Gaj, Tarek El-Ghazawi, Nikitas Alexandridis, Jacek R. Radzikowski, Mohamed Taher, and Frederic Vroman. Effective Utilization and Reconfiguration of Distributed Hardware Resources Using Job Management Systems. In *Proceedings of Parallel and Distributed Processing Symposium, International*, page 8 pp., 22-26 April 2003.

[8] Omar Hassaine. Issues in Selecting a Job Management System. *Sun Blueprint Publications Online*, January 2002. `http://www.sun.com/blueprints/0102/jobsys.pdf`.

[9] HPC-Colony Project. HPC-Colony: Services and Interfaces for Very Large Linux Clusters. `http://www.hpc-colony.org`.

[10] International Business Machines Corporation. IBM LoadLeveler. `http://publib.boulder.ibm.com/clresctr/windows/public/llbooks.html`.

[11] James Patton Jones. Evaluation of Job Queuing/Scheduling Software: Phase 1 Report. Technical Report NAS-96-009, NAS High Performance Processing Group, NASA Ames Research Center, Mail Stop 258-6, Moffett Field, CA 94035-1000, June 1996. `http://www.nas.nasa.gov/Research/Reports/Techreports/1996/PDF/nas-96-009.pdf`.

[12] James Patton Jones. NAS Requirements Checklist for Job Queuing/Scheduling Software. Technical Report NAS-96-003, NAS High Performance Processing Group, NASA Ames Research Center, Mail Stop 258-6, Moffett Field, CA 94035-1000, April 1996. `http://www.nas.nasa.gov/Research/Reports/Techreports/1996/PDF/nas-96-003.pdf`.

[13] James Patton Jones and Cristy Brickell. Second Evaluation of Job Queuing/Scheduling Software: Phase 1 Report. Technical Report NAS-97-013, NAS High Performance Processing Group, NASA Ames Research Center, Mail Stop 258-6, Moffett Field, CA 94035-

1000, June 1997. http://www.nas.nasa.gov/Research/Reports/Techreports/1997/PDF/nas-97-013.pdf.

[14] Joseph A Kaplan. and Michael L. Nelson. A Comparison of Queueing, Cluster and Distributed Computing Systems. Technical report, NASA Langley Research Center, June 1994. http://techreports.larc.nasa.gov/ltrs/PDF/tm109025.pdf.

[15] Sun Microsystems. Sun HPC ClusterTools 5 Software. http://www.sun.com/servers/hpc/software/overview.html.

[16] Platform Computing. Platform Load Sharing Facility (LSF). http://www.platform.com/products/LSF/.

[17] William Saphir, Leigh Ann Tanner, and Bernard Traversat. Job Management Requirements for NAS Parallel System and Clusters. Technical Report NAS-95-006, NAS Scientific Computing Branch, NASA Ames Research Center, Mail Stop 258-6, Moffett Field, CA 94035-1000, February 1995. http://www.nas.nasa.gov/Research/Reports/Techreports/1995/PDF/nas-95-006.pdf.

[18] Steve Sistare. A New Open Resource Management Architecture in the Sun HPC ClusterTools Environment. *Sun Blueprint Online*, November 2002. http://www.sun.com/blueprints/1102/817-0861.pdf.

[19] Sun Microsystems. Sun Grid Engine. http://gridengine.sunsource.net/.

[20] Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny. Condor – A Distributed Job Scheduler. In Thomas Sterling, editor, *Beowulf Cluster Computing with Linux*. MIT Press, October 2001.

[21] Fang Wang, Hubertus Franke, Marios Papaefthymiou, Pratap Pattnaik, Larry Rudolph, and Mark S. Squillante. A gang scheduling design for multiprogrammed parallel computing environments. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 111–125. Springer-Verlag, 1996. Lect. Notes Comput. Sci. vol. 1162.