Libra: An Economy-Driven Cluster Scheduler
Software Requirements Specification

Version <1.0>

Libra: An Economy-Driven Cluster Scheduler	Version: <1.0>
Software Requirements Specification	Date: <23/Dec/01>
First draft	

Revision History

Date	Version	Description	People
<23/Dec/01>	<1.0>	First draft	Project Owner and Client: Rajkumar Buyya Faculty Advisor: Dr. Arif Zaman Project Group: Jahanzeb Sherwani, Nosheen Ali, Nausheen Lotia, Zahra Hayat

Table of Contents

Software Requirements Specification

1. Introduction

1.1 Purpose

This SRS describes the requirements and specifications of Libra, an Economy-Driven Cluster Scheduler. It explains the functional features of the scheduler, along with interface details, design constraints and related considerations such as performance characteristics. The SRS is intended for users and owners of high-performance clusters, cluster management software, job schedulers and grid resources.

1.2 Scope

The Libra Scheduler is intended to work as an add-on to the queuing, scheduling and resource managing module of the open source Sun Grid Engine cluster management system, previously known as Codine. The scheduler will offer market-based economy driven service for managing batch jobs on clusters by scheduling CPU time according to user utility rather than system performance considerations. Hence, the main objective of Libra is to provide Quality of Service (QoS) computational economy in cluster computing.

As part of the project, the Libra Scheduler will only manage sequential and embarrassingly parallel batch jobs to be run on a homogenous Linux cluster, although its functionality can be adapted to schedule resources other than CPU time, even on a heterogeneous cluster.

To provide QoS to users, there will be no mechanism for users to interact with each other, and bargain on the use of resources according to their considerations, as is provided in a grid-computing environment by projects like Nimrod/G [1].

1.3 Definitions, Acronyms and Abbreviations

1.3.1 Cluster/Cluster Computing

Clustering involves connecting two or more computers together to take advantage of combined computational power and resources. Hence, a cluster works as an integrated collection of resources that can provide a single system image spanning all its nodes. Clustering is a popular strategy for processing applications because it transparently spreads the processing of different jobs throughout the cluster, and are used for high-performance applications such as AI expert systems, nuclear simulations, and scientific calculations.

1.3.2 QoS

Quality of Service – meeting what users actually want, thus maximizing their utility.

1.3.3 *Computational Economy* [2]

Computational economy refers to the inclusion of user-specified QoS parameters with jobs so that resource management is based on a user-centric approach rather than on a system-centric approach. This essentially means that user constraints such as deadline and budget are more important in determining the priority of a job by the scheduler, than system policies like ordering jobs according to the basis of submission time. Currently, there is no holistic scheduling mechanism in cluster computing to enable differing QoS levels for different clients.

1.3.4 *Linux*

Linux is an open source operating system, with extensive documentation and user support.

1.3.5 *Grid Computing* [3]

A grid is an infrastructure that couples resources such as computers (PCs, workstations, clusters and so on), software (for special purpose applications) and devices (printers, scanners) across the Internet and presents them as a unified integrated single resource that can be widely used. Grid computing refers to using a grid for processing jobs and processes.

1.3.6 *Bid-based Proportional Resource-Sharing Model* [4]

This is an economic model that specifies one way of allocating (cluster) resources between competing users based on what their criteria is, such as deadline and budget. In this model, the amount of resource allocated to users is proportional to the values of their bids. Essentially, in a bid-based system, the users are allocated tokens or credits, which they can use for having access to resources. The value of each credit depends on the resource demand and cost at the time of usage. We can modify this approach for our cluster, by evaluating this credit/priority according to the deadline and price constraints of users, and considering the system load and resources use as well. This priority is then used by a scheduling policy to allocate resource proportionally to the competing jobs.

1.3.7 *Stride Scheduling* [5]

This is a scheduling algorithm to enforce resource allocations proportional to a user's priority or share. A pool of tickets represents resource rights to a shared resource. A stride is inversely proportional to the number of tickets allocated to a job, hence representing the interval (in terms of CPU time) between two successive scheduling of a job. Jobs with more tickets have shorter strides and hence are scheduled more frequently and obtain a greater fraction of the resource. The algorithm is deterministic and quite accurately schedules jobs in exact proportions based on their ticket allocations.

1.3.8 *Process Migration*

It is the process of moving a job from one computer system to another without restarting the job from the beginning.

1.3.9 Sequential Jobs

These are jobs that must be run on a single node, and cannot be split into separate jobs.

1.3.10 Embarrassingly Parallel Jobs

These are parallel jobs that are comprised of independent parts that can be split up and performed on separate nodes as if they are sequential jobs, with minimal inter-job communication necessary. This is opposed to real parallel jobs that comprise dependent threads that need to interact with each other during the course of execution.

1.3.11 Job accounting

It is a method of quantifying the system resources used by a job.

1.3.12 GUI

Graphical user interface

1.3.13 CPU

Central processing unit of a computer

1.3.14 SGE

Sun Grid Engine cluster management system

1.3.15 Open-source software

Software for which the code is freely available for use and research

1.3.16 Homogenous cluster

A cluster in which the nodes or workstations have uniform characteristics such as the same memory size and hard disk space. They run the same operating system.

1.4 References

[1] R. Buyya, D. Abramson, and J. Giddy, *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*, HPC ASIA'2000, China, IEEE CS Press, USA, 2000.

[2] R. Buyya, D. Abramson, J. Giddy, *An Economy Driven Resource Management Architecture for Global Computational Power Grids*, The 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), Las Vegas, USA, June 26-29, 2000.

[3] R. Buyya, D. Abramson, and J. Giddy, *An Economy Grid Architecture for Service-Oriented Grid Computing*, 10th IEEE International Heterogeneous Computing Workshop (HCW 2001), with IPDPS 2001, SF, California, USA, April 2001.

[4] Rajkumar Buyya, Heinz Stockinger‡, Jonathan Giddy, and David Abramson, *Economic Models for Management of Resources in Peer-to-Peer and Grid Computing*, Monash University.

Libra: An Economy-Driven Cluster Scheduler	Version: <1.0>
Software Requirements Specification	Date: <23/Dec/01>
First draft	

[5] B.N. Chun and D.E. Culler, *Market-based proportional resource sharing for clusters*. Submitted for publication, September 1999.

1.5 Overview

The rest of the SRS examines the specifications of the Libra Scheduler in detail. Section 2 of the SRS presents the general factors that affect the Libra scheduler and its requirements, such as user characteristics and project constraints. Section 3 outlines the detailed, specific functional, performance, system and other related requirements of the Libra scheduler. Supporting information about appendices is provided in Section 3.

2. Overall Description

2.1 Product Perspective

As mentioned earlier, Libra forms part of a larger system comprising the Sun Grid Engine cluster management system that runs on a Linux-based cluster. SGE is responsible for receiving the jobs submitted by the user and delegating them to Libra for scheduling and placing on appropriate workstations or execution hosts. Libra communicates its decision regarding workstation allocation to the resource manager, which then dispatches the job to the chosen workstation. Once assignment to a workstation has occurred, Libra is responsible for implementing the scheduling policy on the jobs executing at the workstations. No modifications to the Linux kernel are required. The details of various interfaces have been provided in Section 3.9.

2.2 Product Functions

The main purpose of the Libra scheduler is to maximize user utility. Thus the job details submitted by the user will include job prioritization criteria: the allocated budget and the deadline required by the user, enabling the scheduler to maximize CPU utilization while remaining within the constraints imposed by the need to optimize QoS.

The scheduler will allocate jobs based on the job parameters**,** which are job specifications submitted by the user with the job, including:

- Location of the executable and input data sets

- Where standard output is to be placed

- System type

- Maximum length of run (execution time)

- Whether the job is sequential or embarrassingly parallel

However, the Libra scheduler will be QoS driven: it will aim to optimize resource utilization *within* user-imposed constraints: thus, user satisfaction is the primary concern, as opposed to maximizing CPU utilization. Thus, the two job parameters most relevant to the scheduling decisions will be:

- Budget allocated by the user to the process

- Deadline

With support from the Sun Grid Engine, the Libra scheduler should embody the following features and functions:

- Should be able to enforce resource allocations according to user-centric priorities

- Should be dynamic, and not static, which is a necessary implication of the user-centric approach, so that users who need their jobs completed in emergency and are willing to pay a high price for it, are able to get their job done through dynamic reallocation of resources even if the job is submitted later than other jobs or the system is heavily loaded. Hence, the scheduler should be able to change resource limits, priorities, privileges and execution order of the submitted jobs.

- Should be scalable, which means that its performance should not degrade with the addition of nodes and jobs to the cluster

- Should be configurable, and allow for various scheduling policies that can be modified to incorporate QoS parameters

- Should be separable from the cluster management system

- Should provide administrative security

- Should provide at least a basic level of job accounting, to aid in scheduling policies

- Should ideally provide a GUI for all components, such as for users to submit jobs and for administrators to oversee scheduling

A market-based economic model for computational economy needs to be developed for the Libra scheduler, which would be responsible for the pricing and allocation of resources according to user constraints. The model is the bid-based proportional resource-sharing model, which will allocate computational time to jobs according to their share determined by the priority of their jobs according to deadline and budget considerations. To ensure that the actual resource allocation reflects the overall job share, the proportional stride-scheduling algorithm will be used.

2.3 User Characteristics

There are essentially two classes of users for the Libra Scheduler used with the Sun Grid Engine cluster management system: the user who wishes to submit jobs for the cluster and the administrator who oversees scheduling and cluster usage. The user needs to know the exact nature of the submitted job, such as the execution time as well as resources

required, and must possess the technical know-how to use the interface for submitting jobs. The administrator must be an advanced cluster operator, fully qualified in using Linux, SGE, and the Libra Scheduler.

2.4 Constraints

The current constraints on the project are related to the provision of hardware resources to implement and test a high-performance cluster. At present, a network of four Pentium-III workstations, with a 128 Mb RAM, serves as the cluster, with SGE running on top of the Linux operating system. For better performance analysis, a larger number of dedicated workstations would be beneficial.

For testing purposes, a simulating tool needs to be used which may have to be specially designed if a generic one is not readily and freely available.

2.5 Assumptions and dependencies

A number of factors that may affect the requirements specified in the SRS include:
- The workability of the Sun Grid Engine cluster management system modules such as those dealing with process migration with the scheduling policies provided by the Libra Scheduler is assumed.
- A basic module of job accounting and payment considerations will be provided, as they are not the focus of the scheduler.
- Users are assumed to have a fair estimate of job execution times, so that the decision to accept or reject a job is facilitated.

2.6 Apportionment of requirements

Detailed requirements with respect to the economic front-end, that is, the exact costing and pricing policy that will be used to charge users for using the cluster will be specified in a later version. The pricing mechanism will not affect the scheduling policies, as they assume a given cost and budget, and schedule resources accordingly.

Furthermore, at present, only a command line Linux interface or the interface of the Sun Grid Engine cluster management system will be used as the user and administrator interface. A GUI might be added once the basic functionality is implemented.

It also remains to be decided whether a software tool for simulation and testing purposes will be designed or a generic one that is freely available will be used.

3. Specific Requirements

3.1 Functionality

This section is organized by the processes and features encapsulated in the Libra

scheduler. First, three features of the overall cluster management system and its interaction with the user or administrator are described: Submit Job, View Job Status, Delete/Change Job. These are linked to the inputs and working processes of the scheduler. They are followed by a detailed specification of the functionality of the Libra scheduler, which are diagrammatically explained by the Data Flow Diagrams provided in Appendix B. Appendix C provides the Data dictionary which should be used to interpret and understand the Data Flow Diagrams.

3.1.1 Submit Job

3.1.1.1 Introduction

This functional feature deals with the end-user (can also be used by the administrator) and is facilitated by the interface of the Sun Grid Engine cluster management system. Hence, it is linked to the scheduler, but does not interface directly with the scheduler. However, the SGE interface will need to be adapted to include additional submission parameters that a user needs to be submit, which are not already supported by SGE. This may alternately be done through a command-line interface.

3.1.1.2 Inputs

User Information – relevant user data such as name and authentication id that needs to be used to submit a job to the cluster.

Job Information – about the job that the user wants to submit, as well as already pending jobs, to determine whether the cluster can accept more jobs and can cater to the particular user job. The job details are provided in the functional feature Initialize Job.

Cluster Information – Cluster Type and Scheduling mechanism details, Current Load Status, Submission Directions and Criteria.

3.1.1.3 Processing

The cluster decides whether it is even open for job submissions, and whether the user that is trying to submit job is a valid user or not. This decision is different from the Accept/Reject Job feature described later on, which deals with the situation when the cluster is open for job submissions, but will decide on whether to accept or reject a job based on job parameters such as budget provided and deadline specified.

3.1.1.4 Outputs

Boolean – Whether the cluster is accepting submission or not.
Boolean – User verified or not.

3.1.2 View Job Status

3.1.2.1 Introduction

This feature will allow the user or administrator to view details about the job that have been submitted, and the progress of the execution of the job.

3.1.2.2   Inputs

User Id and Password – to ensure that only the appropriate user is able to see the job status

3.1.2.3   Processing

Show the user the available options about the submitted job, and information about the progress of job execution. Process selected options if choices are provided to the user to view different aspects of job information.

3.1.2.4   Outputs

User view of available options about jobs submitted by the user.

3.1.3 Delete/Change Job

3.1.3.1   Introduction

This feature may be available to both the user and the administrator, to cancel a job or under specific circumstances, change a job parameter. Parameters such as deadline and cost are crucial to the job scheduling and would not normally be changeable. However, if the user wishes to specify a different output directory or a delayed deadline, this may be permitted.

3.1.3.2   Inputs

User Id and Password – to ensure that only the appropriate user is able to delete/change the job.

3.1.3.3   Processing

Remove job from the queue and update cluster status. Revise scheduling decisions about resource allocation to pending jobs and submission of new jobs.

3.1.3.4   Outputs

Updated cluster information and scheduling decision – CPU load, node status, queue length, quanta allocated to pending jobs, revised expected finishing deadline of pending jobs. These outputs are facilitated by the functionality of the Sun Grid Engine cluster management system, and are not independently produced by the Libra scheduler.

3.1.4 Initialize Job

3.1.4.1   Introduction

This is a main functional requirement that prepares the scheduler for eventually scheduling and executing a job. The Libra scheduler performs it once a job has been accepted by SGE, with information from the scheduler.

3.1.4.2   Inputs

The information submitted by the user about the job, as well as the job id assigned by SGE.

3.1.4.3 Processing

Two processes are performed as part of this function: the details of the job are retrieved and then these details are set in variables that represent the state of the cluster and its queues. The parsed parameters are passed on to the scheduling module where the job may be accepted or rejected.

3.1.4.4 Outputs

The output comprises the set variables that represent job information submitted with a job. Hence, the exact job details are output: Job details – Job Id, Job type, Standalone Execution time, Location of executable and input data sets, System Type, Budget, Deadline.

3.1.5 Accept/Reject Job

3.1.5.1 Introduction

This is the function that is responsible for determining the possibility of satisfying the user, given his budget and deadline. Hence, the most important inputs are budget, deadline, and execution time, to determine whether the job can be accepted or not.

3.1.5.2 Inputs

Job details – Job Id, Job type, Standalone Execution time, Location of executable and input data sets, System Type, Budget, Deadline.

Cluster information – CPU load, node status, remaining time of pending jobs

3.1.5.3 Processing

Based on the job parameters, the scheduler figures out a priority of the job that reflects the share of CPU time that it deserves and needs according to the specified criteria. The scheduler determines whether the job can be finished by the requisite deadline, given the execution time of the job and the execution status of other pending jobs on various nodes.

3.1.5.4 Outputs

The output includes a boolean value specifying whether the job has been accepted or rejected. A suggested deadline or cost is given to the user, as a precondition for the job to be accepted.

3.1.6 Calculate Scheduling Information

3.1.6.1 Introduction

This is the feature of Libra that decides how the job will be scheduled based on its budget and deadline. It uses the Stride Scheduling Algorithm to enforce resource allocations (CPU cycles) proportional to a user's budget and deadline. It is performed by Libra once it has decided that a job can be accommodated.

3.1.6.2 Inputs

Job details – Job Id, Job type, Standalone Execution time, Location of executable and

input data sets, System Type, Budget, Deadline.

3.1.6.3   Processing

Allocate tickets to a job based on its budget and deadline and the budget and deadlines of all other jobs on the execution host under debate. Calculate the stride and the pass of the job, where the stride is simply the global stride constant divided by the tickets and the pass is initially just the stride.

3.1.6.4   Outputs

Scheduling information – the job's tickets, stride and pass.

3.1.7   Determine Execution Host

3.1.7.1   Introduction

This feature decides on which node and on which queue the job will be placed for execution. It is done on the basis of the job's budget, execution time and deadline.

3.1.7.2   Inputs

Job details – Job Id, Job type, Standalone Execution time, Location of executable and input data sets, System Type, Budget, Deadline.

Cluster Information - CPU Load, Node Status, Remaining Time of Pending Jobs, Available Memory

3.1.7.3   Processing

Libra first looks at the load on each of the hosts and sorts them in ascending order, choosing the host that is least loaded. This ensures that load balancing is taken care of. It then selects the appropriate queue based on whether the job is sequential or embarrassingly parallel, budget, etc…

3.1.7.4   Outputs

The chosen execution host and queue.

3.1.8   Dispatch Job

3.1.8.1   Introduction

This feature takes the job and inserts it into the queue on the execution host that the scheduler had previously decided it is to be placed in. This is a separate feature since the job may be able to wait for a while before it is given CPU time, allowing other jobs to complete or be scheduled earlier if their deadlines are more urgent. Thus, the dispatcher decides when to actually send the job for execution.

3.1.8.2   Inputs

The chosen execution host and queue on which to place the job for execution.

3.1.8.3   Processing

The scheduler sends the job to the appropriate queue and it joins the competition for

resources.

3.1.8.4 Outputs

The scheduled job in its appropriate host and queue.

3.1.9 Update Cluster Status

3.1.9.1 Introduction

Every time a job is scheduled for execution on a host, this feature of Libra updates the cluster status. This action is also performed when a job ends and is removed from the cluster.

3.1.9.2 Inputs

Job details – Job Id, Job type, Standalone Execution time, Location of executable and input data sets, System Type, Budget, Deadline.

Chosen execution host and queue.

3.1.9.3 Processing

Libra first reserves the amount of resources that will be required by the job on the node. It then updates the execution host queue status and sends that information about the additional load on the queue and node under consideration to the central information store on the master host.

3.1.9.4 Outputs

There is no output associated with this feature as it simply performs an update and does not send any results back. It is a trigger that is activated every time a job is scheduled or is deleted from an execution host.

3.1.10 Execute Job

3.1.10.1 Introduction

As long as there are jobs in the queues, this feature is being run. It essentially decides how to time-slice between all the jobs currently being executed. These scheduling decisions are made based on the stride-scheduling algorithm.

3.1.10.2 Inputs

Scheduled Job – job details and the queue and node on which that job is scheduled to run.

Scheduling Information – the job's tickets, stride and pass.

3.1.10.3 Processing

It looks at all the jobs on the cluster and selects the one with the minimum pass value, and allots it a quantum. The pass of that job is then advanced by its stride. In case of a tie, the arbitrary FIFO job ordering is used. This loop is repeated until there are no jobs left to run on a queue.

3.1.10.4   Outputs

Updated Cluster Information - CPU Load, Node Status, Remaining Time of Pending Jobs, Available Memory

3.2   Reliability

3.2.1 Maintenance

The scheduler will not support job migration for the purpose of decreasing resource fragmentation.

3.2.2 Maximum bug rate

There will be a maximum of 1 bug/KLOC.

3.2.3 Maximum Time To Repair

In case of system outage, the cluster will be down while the Linux operating system boots up. This will take less than five minutes.

3.2.4 Security Considerations

The Libra scheduler will ensure the privacy of user job status and ensure full control over job execution, so that alteration of scheduling criteria or actual resource allocation is not possible without administrator authority.

3.3   Performance Requirements

3.3.1 Response time

The maximum response time for the submission of a job will be 1 minute.

3.3.2 Capacity

The maximum number of jobs schedulable is limited only by the capacity of the nodes to fulfill the jobs' deadlines; there is no upper limit inherent in the Libra scheduler as such.

3.3.3 Deadline sensitivity

Assuming submitted statistics for jobs are accurate, the Libra scheduler will ensure that all jobs are completed with a 10% error allowance.

3.3.4 Cost sensitivity

Under all circumstances, the maximum cost payable as submitted by the user will be the maximum cost charged to the user.

3.4 Supportability

3.4.1 Naming Convention

All code will be written as specified by the Hungarian Naming Convention.

3.4.2 Coding Standards

All code will be written as required by the GNU General Purpose License.

3.5 Design Constraints

3.5.1 Parent Component: Sun Grid Engine

The Libra scheduler will be a sub-component of SGE.

3.5.2 Software Language

All coding will be done in standard C.

3.5.3 Scheduling Constraints

Exhaustive searches of the entire set of combinations of jobs will not be done. Heuristics will be developed for this scheduling problem.

3.6 Online User Documentation and Help System Requirements

All documentation will be made in accordance with requirements pertaining to open source software under the GNU General Purpose License. Additionally, on-line user documentation will be in the form of **man** pages accessible through Linux.

3.7 Purchased Components

The entire Libra project is open source, and hence no component of it will be purchased.

3.8 Interfaces

3.8.1 User Interfaces

There will be two user types – the cluster user and the cluster administrator – each of which will have its own corresponding interface.

3.8.1.1 Cluster User Interface

The minimal requirements are that the cluster user would be able to interact with the system through the Linux command prompt, or through the interface provided by the Sun Grid Engine cluster management system. There will be a different command for each of the following actions:

- submit jobs with the associated deadline, cost, and execution time

- query the cluster to establish the current cost per unit time for submitting new jobs

- monitor the status of submitted jobs

- cancel jobs submitted by him

- check his credit balance

- check his usage history

3.8.1.2 Cluster Administrator Interface

The minimal requirements are that the cluster administrator will be able to interact with the system through the Linux command prompt, or through the interface provided by the Sun Grid Engine cluster management system. In addition to the commands available to the user, additional commands will allow the administrator to:

- check the status of each node of the cluster

- check the usage pattern history of the cluster

- check the status of all submitted jobs

- check the load on each node of the cluster

- alter the cost structure of the cluster

- alter the scheduling policy of the cluster

- cancel, suspend, and resume any job

3.8.2 Hardware Interfaces

All hardware interfaces of Libra will be those of the Sun Grid Engine on top of which it will be running. Hence, it will incorporate SGE's interfaces for:

- CPU usage

- Memory usage

- Swap file creation

- Network communication

3.8.3 Software Interfaces

Libra will directly interface with the modified Sun Grid Engine cluster management system, version 5.3. SGE in turn will be interfacing with the operating system and any other software components it requires. For the execution of embarrassingly parallel jobs, the PVM (Parallel Virtual Machine) and MPI (Message Passing Interface) libraries will be used by the cluster.

3.8.4 Communications Interfaces

Libra will utilize the communications architecture of Sun Grid Engine itself, and will not have any unique communications interfaces above of SGE.

3.9 Licensing Requirements

Libra will be released under a GPL license and will be open-source.

3.10 Legal, Copyright and Other Notices

Sun Grid Engine (SGE) is a copyright trademark of Sun Microsystems, Inc. Libra Scheduler is a copyright trademark of Mr. Rajkumar Buyya, Monash University. MPI is a copyright of the University of Chicago and Mississippi State University. PVM is a copyright of the University of Tennessee, Oak Ridge National Laboratory, and Emory University.

4. Supporting Information

4.1.1 Appendix A – Background Research (dated 10th Nov, 2001) on:

- Cluster Computing
- Market-based Economic Models
- Cluster Scheduling
- Cluster Management Software

4.1.2 Appendix B – Data Flow Diagrams

4.1.3 Appendix C – Data Dictionary