# Libra: An Economy-Driven Job Scheduling System for Clusters

**Jahanzeb Sherwani, Nosheen Ali, Nausheen Lotia, Zahra Hayat**
Lahore University of Management Sciences
Sector U, DHA, Lahore, Pakistan
jahanzeb@cmu.edu

**Rajkumar Buyya**
Grid Computing and Distributed Systems Lab.
Dept. of Computer Science and Software Engg.
The University of Melbourne, Australia
raj@cs.mu.oz.au

## ABSTRACT
Clusters of computers have emerged as mainstream parallel and distributed platforms for high-performance, high-throughput and high-availability computing. To enable effective resource management on clusters, numerous cluster managements systems and schedulers have been designed. However, their focus has essentially been on maximizing CPU performance, but not on improving the value of utility delivered to the user and quality of services. This paper presents a new computational economy driven scheduling system called Libra, which has been designed to support allocation of resources based on the users' quality of service (QoS) requirements. It is intended to work as an add-on to the existing queuing and resource management system. The first version has been implemented as a plugin scheduler to the PBS (Portable Batch System) system. The scheduler offers market-based economy driven service for managing batch jobs on clusters by scheduling CPU time according to user-perceived value (utility), determined by their budget and deadline rather than system performance considerations. The Libra scheduler ensures that both these constraints are met within an O(n) run-time. The Libra scheduler has been simulated using the GridSim toolkit to carry out a detailed performance analysis. Results show that the deadline and budget based proportional resource allocation strategy improves the utility of the system and user satisfaction as compared to system-centric scheduling strategies.

## Keywords
Cluster computing, scheduling, economy-driven, user value

## INTRODUCTION
Clusters of commodity computers (PCs) have emerged as mainstream parallel and distributed platforms for high-performance, high-throughput and high-availability computing [1]. They are presented together as a single, unified resource to the end users by middleware technologies such as resource management and scheduling system. Users submit jobs in batch to a resource management system queue and a centralized scheduler decides how to prioritize and allocate resources for jobs execution. To minimize the response time, the scheduling system strategy needs to prioritize competing user jobs with varying levels of priorities and importance and allocate resources accordingly. To perform these tasks effectively, schedulers require knowledge of how users value their computations and their quality of service requirements, which varies from time to time [6]; schedulers should be able to provide a feedback signal that prevents users from submitting unbounded amounts of work [8]. Unfortunately, the current approaches (e.g., Condor [2], PBS [9], SGE [11], and LSF [12]) to batch scheduling provide limited means for users to expresses their valuation of resources and quality of service requirements, if any. Also, feedback signals provide no incentive for users to pay attention to and respond to them.

To overcome the shortcomings of the traditional system-centric cluster management systems, we advocate the use of the computational economy metaphor [4][13][14][15]. Computational economy enables the regulation of supply and demand of resources. It allows the users to specify QoS parameters with jobs and also offers incentive to users for relaxing QoS requirements. This essentially means that user constraints such as deadline and budget are more important in determining the priority of a job by the scheduler rather than system policies such as ordering jobs according to the basis of submission time.

## RELATED WORK AND TECHNOLOGIES
In our background research, we came across the following related work and technologies that have either influenced or have served as a base platform during the implementation and evaluation of Libra scheduler: REXEC Cluster Scheduler [7], Portable Batch System [9], Nimrod-G Grid Resource Broker [3], GridSim Toolkit [6], and PBSWeb [10].

## RESOURCE MANAGEMENT AND SCHEDULING (RMS) THROUGH LIBRA
Resource Management and Scheduling (RMS) is the act of distributing applications among computers to maximize their throughput. It also enables the effective and efficient utilization of the resources available. The software that performs the RMS consists of two components: a resource manager and a resource scheduler. The resource manager component is concerned with problems, such as locating and allocating computational resources, authentication, as well as tasks such as process creation and migration. The

resource scheduler component is concerned with tasks such as queuing applications, as well as resource location and assignment.

The Libra RMS architecture consists of clients, a server (that acts as mediator between the user and cluster nodes), and a set of cluster nodes that provide the computing horse power. Each cluster node runs a resource-monitoring daemon that maintains up-to-date information about the node in which it resides. A user interacts with the RMS environment via a client program, which could be a Web browser or a customized X-windows interface, through which job details may be provided to the server. These details may include information such as location of the executable and input data sets, where standard output is to be placed, system type, maximum length of run, whether the job needs sequential or parallel resources, and so on. In addition, Libra allows the users to express valuation of computations and QoS requirements via the parameters such as deadline and budget. Once a job has been submitted to Libra, it uses the job details and users' QoS requirements to place, schedule, and run the job.

## LIBRA SCHEDULING ALGORITHM

When users submit jobs, they must submit them with values for the job's estimated run-time (assuming a stand-alone node), the deadline within which they require results, and the budget they are willing to pay to get the job done within this deadline. These three variables will be referred to by E, D, and B, respectively. The server receives such requests, and based on criteria to be described, either accepts or rejects the job (informing the user in eithercase).

The assumptions of the algorithm are as follows:

1) All cluster nodes are dedicated (i.e., they run only jobs approved by the server), and heterogenous (i.e., having the same hardware and software running on them).

2) Further, the underlying operating system on the cluster nodes accepts a parameter that is the percentage of CPU the job must be allocated, and must be able to enforce this value (as long as the sum of all shares is less than or equal to 100).

3) The estimated runtime (E) given by the user is correct for a standalone job running on any node of the cluster.

Thus, users submit jobs with these three values to the central server (the gateway into the cluster). There is 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.

The server first checks whether the budget is acceptable based on a simple minimum cost formula. The formula we developed was:

Cost = $\alpha$*E + $\beta$*E/D (where $\alpha$ and $\beta$ are coefficients)

The value of $\alpha$ has impact on the resource pricing/job processing cost, which can be driven by demand and supply for resources. The value of $\beta$ has impact on the incentive offered to the user for specifying the actual deadline. It offers higher incentive for users for relaxing the deadline.

The logic of this cost function is as follows. For a specific job, the user must be charged on the amount of cluster hours he is using regardless of deadline – thus, for longer jobs, he will be charged more than for shorter ones. This is managed by the first term in the equation. The second term is a measure of the user's 'niceness' – the relative sacrifice he is giving in terms of his ratio of estimate to deadline. For a constant estimate, if the user increases his deadline, he is allowing the cluster more time to handle his request, and hence, should be charged less than if he were giving the cluster less time (and hence, requesting resources more urgently). The sum of both these values is the actual cost to the user.

If the budget submitted by the user is acceptable, each of the cluster's nodes is queried to see whether they can complete this job within its deadline (this decision-making is explained below). The response by each node is either a rejection, or an acceptance, in which case the node also gives a measure of the current load it is currently servicing. Thus, if more than one node returns an acceptance, the least loaded node is the one which will finally be sent the job.

Each node keeps track of the status of jobs running on it; specifically, the CPU-time they have run for so far, and hence, the remaining run-time for the job. Additionally, nodes also keep track of the time left for the job's deadline to expire. With these two values (run-time left, and deadline left), the nodes can calculate the required CPU share that needs to be dedicated to the job so that it may complete its remaining CPU-hours within the remaining deadline. For instance, if a job has a remaining run-time of 2 CPU-hours, but a deadline of 4 realtime-hours, the node needs to allocate at least 50% of the CPU to the job over the next 4 hours for it to finish within the deadline.

Assuming it accepts the newly requested job, the node calculates the total requirement of the CPU by all of its jobs. If and only if the summation of all jobs' CPU requirements is less than or equal to 100%, the job is acceptable, and the node informs the server of this. Also, the node sends the server the value of total CPU percentage as it just calculated (which is the required load on the node if the newly requested job is sent to this node).

If none of the nodes can run the job, the server informs the user that its deadline could not be met by any node, and hence the user must try later, or try again with a more relaxed deadline. However, if there is one or more node that can run the job, the job is sent to the least loaded node which responded with an acceptance. The node receives the job, and dispatches it, along with the CPU share requirement, to the operating system. Periodically (depending on the CMS and the OS), the node receives a value of CPU-time completed from the OS, and updates its internal book-keeping with this new value. If the job has run more or less than it was supposed to in this time, the node may request the OS to update its CPU share with a new value as calculated taking into account the new values of run-time left and deadline left.

Thus, by ensuring that each node is keeping within the CPU time requirements, the Libra scheduler is able to guarantee that jobs will be completed within their deadline; if any new job is to be dispatched to any node, it will only be allowed if, by virtue of its load requirements, its load requirements are possible to be met along with the other jobs running on the node.

## IMPLEMENTATION

The implementation and evaluation of Libra scheduler and its algorithm is achieved by leveraging existing software infrastructure. The Libra scheduler has been implemented as an add-on package for PBS resource management system. The PBSWeb interface has been enhanced to support Libra's mechanism to allow users to express their valuation of computations and QoS requirements. The performance of the scheduling algorithm has been evaluated through simulation using the GridSim toolkit.

### Libra with PBS

For actual performance of the Libra algorithm, we implemented Libra in Portable Batch System (PBS). The implementation tested successfully on a cluster of 4 nodes, with job acception, rejection, and completion easily repeatable and verifiable.

### Libra with PBSWeb

For making our cluster more user-friendly and accessible, we designed an efficient front-end for the PBS-Libra engine using PBSWeb, an interface add-on to PBS developed at the University of Alberta. The PBSWeb interface was modified to work with Libra, included the provision for E, D and B with the job parameters.

### Libra Simulation Using the GridSim Toolkit

For simulation purposes we looked to the GridSim toolkit that allows modeling and simulation of parallel and distributed system entities for evaluation of scheduling algorithms. Although, it has been developed to simulate scheduling on grids (multiple users competing for heterogeneous resources), we have been able to use it simulate scheduling on clusters with minimal effort.

## PERFORMANCE EVALUATION

The majority of the evaluation was carried out using GridSim. The only result that was sufficiently different from what we expected was that two jobs (carrying out mathematically intensive graphical ray-tracing) running simultaneously on a node ran faster when timeshared than when run sequentially one after the other. Our assumption was that timesharing increases overhead via context switching; however, we hypothesize that the offsetting factor is that timesharing increases the throughput of the CPU because each job can use a different resource simultaneously (e.g., disk I/O along with math calculations). Generating 2 batches of jobs (100 jobs, 200 jobs) of different sizes with deadlines and budgets, we ran the same data on clusters of 10 and 20 nodes and compared the results obtained using two different scheduling policies: Libra and FIFO (First In First Out).

The 1st batch with 100 jobs/experiments is formulated as follows. Arrival times were spread randomly over the experiments, ranging from t=1 to t=102 simulation time. Length of the jobs ranged randomly from 1000 MIs to 10900 MIs. 80% of the jobs were with a budget of b=1000, the other 20% had random amounts ranging from b=1000 to b=12000. Deadlines again ranged randomly from d=1 to d=1200.

The 2nd batch with 200 jobs/experiments is formulated as follows. Arrival times were spread randomly over the experiments, ranging from t=1 to t=208 simulation time. Length of the jobs ranged randomly from 1000 MIs to 10900 MIs. 80% of the jobs were with a budget of b=1000, the other 40% had random amounts ranging from b=1000 to b=12000. Deadlines again ranged randomly from d=1 to d=1200.

The results based on applying these input data to both Libra and PBS' default FIFO scheduling policy are summarized in Table 1. Job completion in Libra implies that the job was first accepted and then completed within the deadline; if the job was rejected, it counts as not having completed within the deadline. Job completion in PBS's FIFO is simply whether or not the job managed to complete within the user's given deadline, as there is no job admission control in FIFO's policy (hence, no possibility of job acceptance/rejection), and no sensitivity to user deadline.

| No. of Jobs | No. of Nodes | No. of Jobs Completed in Time | | No. of Jobs Not Completed in Time | |
|---|---|---|---|---|---|
| | | PBS FIFO | Libra | PBS FIFO | Libra |
| 100 | 10 | 77 | 86 | 23 | 14 |
| | 20 | 86 | 90 | 14 | 10 |
| 200 | 10 | 95 | 102 | 105 | 98 |
| | 20 | 165 | 177 | 35 | 23 |

Table 1: No. of jobs accepted and rejected by PBS FIFO and Libra scheduling strategies.

The results showed that in all four cases the Libra proportional share algorithm performs better than the PBS's FIFO algorithm, in that it completed more jobs within the deadline. Although there isn't much of a difference between the two algorithms when the cluster is large and the workload is small, this does not mean that this is always the case. We see that as the workload increases the difference between the number of jobs successfully completed between the two increases.

## CONCLUSION AND FUTURE WORK

There is a great scope for economy-based scheduling in HPC as HPC resources are shared among many users. Hence, the allocation of resources based on the amount they pay and the deadline within which they require results is more relevant to them. Thus, architectures that incorporate such constraints would optimize user utility by completing jobs within this budget and deadline.

The algorithm we developed can be improved much further. A simple extension could be that instead of time-sharing resources on one node, each node should be dedicated to each job (with more urgent jobs pre-empting already-running jobs if by doing so more jobs can be handled within their deadlines), and all calculations of whether jobs can be finished within their deadlines should be carried out with that in mind. This model benefits from minimized context switching; however, it disallows the simultaneous use of different resources by timeshared jobs. In addition, the entire structure of user-based parameters needs to be tested 'in the wild', with real users submitting jobs on such a system. This would aid in fine-tuning the current system to users' requirements, as well as in discovering new user parameters, unaccounted for in Libra.

Future work can focus on developing new deadline and budget driven algorithms for scheduling applications created using the parameter-sweep model (where one program is to be run repeatedly using different parameters every time), and parallel applications. In addition, different pricing strategies driven by supply-and-demand for resources and economic models need to be investigated.

## ACKNOWLEDGEMENTS

Special thanks to the members of the PBS mailing list who gave us answers to many of our questions. We thank Shoaib Burq and Srikumar Venugopal for their comments.

Further information on Libra and its source code is available from the Gridbus Project: http://www.gridbus.org

## REFERENCES

1. R. Buyya (ed.). *High Performance Cluster Computing: Architectures and Systems*, Volume 1, Prentice Hall, USA, 1999.
2. J. Basney and M. Livny, Deploying a High Throughput Computing Cluster, *High Performance Cluster Computing*, R. Buyya, Editor, Vol. 1, Chapter 5, Prentice Hall PTR, May 1999.
3. R. Buyya, D. Abramson, and J. Giddy, Nimrod-G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid, *Proceedings of the 4th International Conference and Exhibition on High Performance Computing in Asia-Pacific Region (HPC ASIA 2000)*, May 14-17, 2000, Beijing, China.
4. R. Buyya, D. Abramson, and J. Giddy, An Economy Driven Resource Management Architecture for Global Computational Power Grids, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, June 26-29, 2000, Las Vegas, USA, 2000.
5. R. Buyya and M. Murshed, GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing, *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, Wiley Press, May 2002.
6. R. Buyya, D. Abramson, and J. Giddy, An Economy Grid Architecture for Service-Oriented Grid Computing, *10th IEEE International Heterogeneous Computing Workshop (HCW 2001)*, California, USA, April 2001.
7. B. Chun and D. Culler, REXEC: A Decentralized, Secure Remote Execution Environment for Clusters, *Proceedings of 4th Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing*, Toulouse, France, January 2000.
8. B. Chun and D. Culler, User-centric Performance Analysis of Market-based Cluster Batch Schedulers, *Proceedings of 2nd IEEE International Symposium on Cluster Computing and the Grid, Berlin (CCGrid 2002)*, Germany, May 2002.
9. Veridian Systems, OpenPBS v2.3: The Portable Batch System Software, Veridian Systems, Inc., CA, Sept. 2000. http://www.openpbs.org/scheduler.html
10. George Ma and Paul Lu. PBSWeb:A Web-based Interface to the Portable Batch System, *12th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, Las Vegas, U.S.A., November 6-9, 2000.
11. W. Gentzsch, Sun Grid Engine (SGE): A Cluster Resource Manager, http://gridengine.sunsource.net/
12. Platform, Load Sharing Facility (LSF), http://www.platform.com/products/wm/LSF/
13. C. Waldspurger, T. Hogg, B. Huberman, J. Kephart, and W. Stornetta, Spawn: A Distributed Computational Economy, *IEEE Transactions on Software Engineering*, Vol. 18, No. 2, IEEE CS Press, USA, February 1992.
14. M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin, An Economic Paradigm for Query Processing and Data Migration in Mariposa, *Proceedings of 3rd International Conference on Parallel and Distributed Information Systems,* Austin, USA, 28-30 Sept. 1994, IEEE CS Press, 1994.
15. B. Chun and D. Culler, Market-based Proportional Resource Sharing for Clusters, University of California at Berkeley, Computer Science Division, *Technical Report CSD-1092*, January 2000.