Pricing for Utility-driven Resource Management and Allocation in Clusters

Chee Shin Yeo and Rajkumar Buyya
Grid Computing and Distributed Systems Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne
VIC 3010, Australia
{csyeo, raj}@csse.unimelb.edu.au

Abstract

Users perceive varying levels of utility for each different job completed by the cluster. Therefore, there is a need for existing cluster Resource Management Systems (RMS) to provide a means for the user to express its perceived utility during job submission. The cluster RMS can then obtain and consider these user-centric needs such as Quality-Of-Service requirements in order to achieve utility-driven resource management and allocation. We advocate the use of *computational economy* for this purpose. In this paper, we describe an architectural framework for a utility-driven cluster RMS. We present a user-level job submission specification for soliciting user-centric information that is used by the cluster RMS for making better resource allocation decisions. In addition, we propose a dynamic pricing function that the cluster owner can use to determine the level of sharing within a cluster. Finally, we define two user-centric performance evaluation metrics: *Job QoS Satisfaction* and *Cluster Profitability* for measuring the effectiveness of the proposed pricing function in realizing utility-driven resource management and allocation.

I. Introduction

Cluster computing [1][2] is increasingly used for high-performance, high-throughput and high-availability computing in a wide variety of application areas. Clusters are not only used for executing compute-intensive applications, but also as replicated storage and backup servers that provide the essential fault tolerance and reliability for critical applications.

Mission critical cluster middlewares create the Single System Image (SSI) [3] that presents a single unified computing resource to the user. This provides better usability and transparency for the users as it hides the complexities of the underlying distributed and heterogeneous nature of clusters from them. An example of such a middleware is the *cluster Resource Management System* (RMS) that provides a single interface for user-level sequential and parallel applications to be executed on the cluster.

For effective and efficient management, the cluster RMS requires knowledge of how users value the resources that are being competed for [4] and having a feedback mechanism that prevents users from submitting unlimited quantities of work [5]. However, existing cluster RMSs provide no or minimal support for users to define Quality of Service (QoS) requirements during job submission. For instance, the user cannot specify the deadline when the job should finish execution and the budget that he is willing to pay for the execution before the deadline. They continue to use system-centric approaches that focus on increasing the throughput and maximizing the utilization of the cluster. They neglect the need to use utility models for allocation and management of resources that would otherwise consider and thus able to achieve the users' desired utility.

We advocate the use of *computational economy* [6][7][8][9][10][11] for achieving utility-driven resource management and allocation in clusters since system-centric management for shared resources is not effective due to lack of economic accountability. Computational economy is able to regulate supply and demand of cluster resources at market equilibrium, provides feedback in terms of economic incentives for both users and cluster owner, and promotes QoS-based resource allocation that caters to users' needs.

This paper focuses on a pricing mechanism to support utility-driven management and allocation of resources in a cluster. First, the architecture of existing cluster RMS that uses system-centric approaches is extended to adopt economy-based resource management and allocation. A simple and extensible user-level job submission specification provides a means for users to specify user-centric information such as resource and QoS requirements. Economy-based mechanisms then make use of this information and incorporate a pricing function to enforce resource allocations. The effectiveness of the economy-based mechanisms is examined using two user-centric evaluation metrics: *Job QoS Satisfaction* and *Cluster Profitability*.

The rest of this paper is organized as follows. Section II discusses related work. Section III presents an architectural framework for a utility-driven cluster RMS. Section IV describes the user-level job submission specification for soliciting user-centric information for each job. Section V defines a pricing function that satisfies four essential requirements for pricing cluster resources. Section VI outlines the admission control, resource allocation, and job control mechanisms that together enforce the utility to be achieved by the cluster. Section VII discusses performance evaluation results of the proposed pricing function using two user-centric evaluation metrics and Section VIII concludes this paper.

II. Related Work

There are a number of cluster RMSs such as Condor [12], LoadLeveler [13], Load Sharing Facility (LSF) [14], Portable Batch System (PBS) [15], and Sun Grid Engine (SGE) [16]. But, these existing Cluster RMSs adopt system-centric approaches that optimize overall cluster performance. For example, the cluster RMS aims to maximize processor throughput and utilization for the cluster, and minimize average waiting time and response time for the jobs. But, these system-centric approaches neglect and thus ignore user-centric required services that truly determine users' needs and utility. There are no or minimal means for users to define QoS requirements and their valuations during job submission so that the cluster RMS can improve the value of utility. We propose an architectural framework for extending these existing cluster RMSs to support utility-driven resource management and allocation, and describes how economy-based mechanisms can be incorporated to achieve that.

Maui [17] is an advanced scheduler that supports configurable job prioritization, fairness policies and scheduling policies to maximize resource utilization and minimize job response time. It provides extensive options for the administrator to configure and define various priorities of jobs to determine how resources are allocated to jobs. Maui also allows users to define QoS parameters for jobs that will then be granted additional privileges and supports advance reservation of resources where a set of resources can be reserved for specific jobs at a particular timeframe. In addition, Maui can be integrated as the scheduler for traditional cluster RMS such as Loadleveler, LSF, PBS and SGE. But, Maui does not provide economic incentives for users to submit jobs with lower priority or QoS requirements and cluster owner to share resources.

REXEC [10] is a remote execution environment for a campus-wide network of workstations, which forms part of the Berkeley Millennium Project. REXEC allows the user to specify the maximum rate (credits per minute) that he is willing to pay for processor time. The REXEC client then selects a compute node that matches the user requirements and executes the application directly on it. It uses a proportional resource allocation mechanism that allocates resources to jobs proportional to the user valuation irrespective of their job needs. However, our economy-based resource allocation mechanism prioritizes and allocates resources to jobs based on the QoS needs of each job. We allocate resources proportionally to jobs with respect to their required QoS such as deadline rather than user valuation so that more jobs are completed with their QoS fulfilled.

Libra [11] is an initial work done that successfully demonstrates that an economy-based scheduler is able to deliver more utility to users compared to traditional scheduling policies. Libra allows users to specify QoS requirements and allocates resources to jobs proportional to their specified QoS requirements. Thus, Libra can complete more jobs with their QoS requirements satisfied as compared to system-centric scheduling policies that do not consider various QoS needs of different jobs. Currently, Libra computes a static cost that provides incentives for jobs with a more relaxed deadline so as to encourage users to submit jobs with a longer deadline. But, Libra does not consider the actual supply and demand of resources, thus users can continue to submit unlimited amount of jobs into the cluster if they have the budget. In this paper, we propose an enhanced pricing function that satisfies four essential requirements for pricing of cluster resources and prevents the cluster from overloading.

III. Architectural Framework

We describe an architectural framework for extending an existing system-centric cluster RMS to support utility-driven resource management and allocation. Fig. 1 shows the architectural framework for a utility-driven cluster RMS. Four additional mechanisms: Pricing, Economy-based Admission Control, Economy-based Resource Allocation, and Job Control (shaded in Fig. 1) are to be implemented as pluggable components into the existing cluster RMS architecture to support utility-driven resource management.

A utility-driven cluster RMS needs to determine the cost the user has to pay for executing a job and fulfilling his QoS requirements. This in turn generates economic benefits for the cluster owner to share the cluster resources. We propose a *Pricing* mechanism that employs some pricing function for this purpose. Later in this paper, we discuss a pricing function that aims to be flexible, fair, dynamic and adaptive.

There should also be an admission control mechanism to control the number of jobs accepted into the cluster. If no admission control is implemented, increasing job submissions will result in fewer jobs to be completed with the required QoS due to insufficient cluster resources for too many jobs. We propose an *Economy-based Admission Control* mechanism that uses dynamic and adaptive pricing (determined by the Pricing mechanism) as a natural means for admission control. For example, increasing demand of a particular resource increases its price so that fewer jobs that have sufficiently high budget will be accepted. In addition, our Economy-based Admission Control mechanism also examines the required QoS of submitted jobs to admit only jobs whose QoS can be satisfied.

After a job is accepted, the cluster RMS needs to determine which compute node can execute the job. In addition, if there are multiple jobs waiting to be allocated, the cluster RMS needs to determine which job has the highest priority and should be allocated first. We propose an *Economy-based Resource Allocation* mechanism that considers user-centric requirements of jobs such as required resources and QoS parameters like deadline and budget, and allocate resources accordingly to these needs.

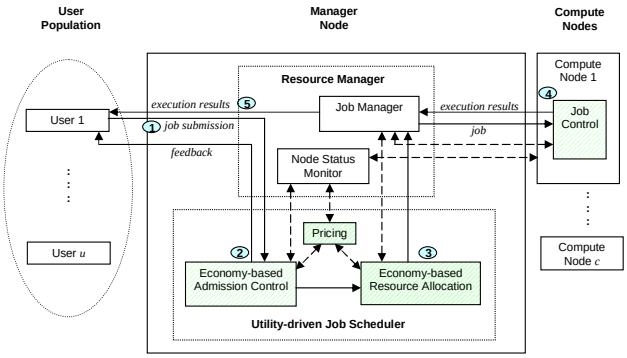


Fig. 1. Architectural framework for a utility-driven cluster RMS. The Economy-based Admission Control mechanism determines whether a job submitted into the cluster should be accepted or rejected and feedback to the user. If accepted, the Economy-based Resource Allocation mechanism determines the best compute node to execute the job. The Job Control mechanism then enforces the resource allocation to ensure that the required utility is achieved.

After resource allocation, there should be a mechanism to enforce the resource allocation so as to ensure that the required level of utility can be achieved. We propose a *Job Control* mechanism at each compute node that monitors and adjusts the resource allocation if necessary.

As shown in Fig. 1, there are u local users who can submit jobs to the cluster for execution. The cluster has a single manager node and c compute nodes. The centralized resource manager of the cluster RMS is installed on the manager node to provide the user interface for users to submit jobs into the cluster. The typical flow of a job in a utility-driven cluster RMS (circled numbers in Fig. 1) is as follows:

1) A user submits a job to the cluster RMS using the user-level job submission specification.
2) The Economy-based Admission Control mechanism determines whether the job should be accepted or rejected based on the job details and QoS requirements given in the job submission specification and current workload commitments of the cluster. The outcome is feedback to the user.
3) If the job is accepted, the Economy-based Resource Allocation mechanism determines which compute node the job is to be allocated to. The job manager is then informed to dispatch the job to the selected compute node.
4) The Job Control mechanism administers the execution of the job and enforces the resource allocation.
5) The job finishes execution and its execution result is returned to the user.

IV. User-level Job Submission Specification

We propose a simple generic user-level job submission specification to capture user-centric information defined as follows:

$$job_i([Segment_1][Segment_2]...[Segment_s]) \tag{1}$$

Each job i submitted to the cluster has a corresponding submission specification comprising of s segments. Each segment acts as a category that contains fine-grain parameters to describe a particular aspect of job i.

The job submission specification is designed to be extensible such that new segments can be added into the specification and new parameters can be added within each segment. This flexibility can thus allow customization for gathering varying information of jobs belonging to different application models. For instance, a job belonging to a workflow-based application may have a data dependency segment.

Currently, we identify a basic job submission specification that consists of four segments for a parallel compute-intensive job:

$$\begin{aligned} job_i(\ &[JobDetails][ResourceRequirements] \\ &[QoSConstraints][QoSOptimization]) \end{aligned} \tag{2}$$

The first segment, $JobDetails$ describes information pertaining to the job. This provides the cluster RMS with necessary knowledge that may be utilized for more effective resource allocation. One basic example of $JobDetails$ is:

1) Runtime Estimate RE_i: the estimated time needed to complete a job i on a compute node. We define the runtime R_i of job i as the time period for it to be processed on a compute node provided that it is allocated the node's full proportion of processing power. Thus, the runtime varies on nodes of different hardware and software architecture and does not include any waiting time and communication latency. The runtime can also be expressed in terms of the job length in million instructions (MI).

The second segment, $ResourceRequirements$ specifies the resources that are needed by the job in order to be executed on a compute node. This facilitates the cluster RMS to determine whether a compute node has the necessary resources to execute the job. Two basic examples of $ResourceRequirements$ are:

1) Memory size MEM_i: the amount of local physical memory space needed to execute job i.
2) Disk storage size $DISK_i$: the amount of local hard disk space required to store job i.
3) Number of processors $PROC_i$: the number of processors required by job i to run.

The third segment, $QoSConstraints$ states the QoS characteristics that have to be fulfilled by the cluster RMS. This captures user-centric requirements that are necessary to achieve the user's perceived utility. Two basic examples of $QoSConstraints$ are:

1) Deadline D_i: the time period in which job i has to be finished.
2) Budget B_i: the budget that the user is willing to pay for job i to be completed with the required QoS (eg. deadline) satisfied.

The fourth segment, $QoSOptimization$ identifies which QoS characteristics to optimize. This supports user personalization whereby the user can determine specific QoS characteristics he wants to optimize. Two basic examples of $QoSOptimization$ are:

1) Finish time FT_i: the time when job i finishes execution on a compute node. This means that the user wants the job to be finished in the shortest time, but within the specified budget.
2) Cost C_i: the actual cost the user pays to the cluster for job i provided that the required QoS is satisfied. This means that the user wants to pay the lowest cost for completing the job.

This example for a parallel compute-intensive job demonstrates the flexibility and effectiveness of the proposed generic user-level job submission specification in soliciting user-centric requirements for different application models. Users are able to express their job-specific needs and desired services that are to be fulfilled by the cluster RMS for each different job. The cluster RMS can utilize these information to determine which jobs have higher priority and allocate resources accordingly so as to maximize overall users' perceived utility, thus achieving utility-driven resource management and allocation.

V. Pricing of Resources

A. Four Essential Requirements

We outline four essential requirements for defining a pricing function to price cluster resources. First, the pricing function should be *flexible* so that it can be easily configured by the cluster owner to modify the pricing of resources to determine the level of sharing. Second, the pricing function has to be *fair*. Resources should be priced based on actual usage by the users. This means that users who use more resources pay more than users who use fewer resources. With QoS, users who specify high QoS requirements (such as a short deadline) for using a resource pay more than users who specify low QoS requirements (a long deadline). Third, the pricing function should be *dynamic* such that the price of each resource is not static and changes depending on the cluster operating condition. Fourth, the pricing function needs to be *adaptive* to changing supply and demand of resources so as to compute the relevant prices accordingly. For instance, if demand for a resource is high, the price of the resource should be increased so as to discourage users from overloading this resource and to maintain equilibrium of supply and demand of resources.

B. Pricing Function

We define a pricing function that is able to satisfy the above mentioned four essential requirements for pricing of cluster resources. Examples of cluster resources that are utilized by a job are processor time, memory size and disk storage size. The pricing function computes the pricing rate P_{ij} for per unit of cluster resource utilized by job i on compute node j as:

$$P_{ij} = (\alpha * PBase_j) + (\beta * PUtil_{ij}) \tag{3}$$

The pricing rate P_{ij} comprises of two components: a static component based on the base pricing rate $PBase_j$ for utilizing the resource on compute node j and a dynamic component based on the utilization pricing rate $PUtil_{ij}$ of that resource that takes into account job i. The factors α and β for the static and dynamic components respectively provides the flexibility for the cluster owner to easily configure and modify the weightage of the static and dynamic components on the overall pricing rate P_{ij}.

The cluster owner specifies the fixed base pricing rate $PBase_j$ for per unit of cluster resource. For instance, $PBase_j$ can be \$1 per second for processor time, \$2 per MB for memory size, and \$10 per GB for disk storage size. $PUtil_{ij}$ is computed as a factor of $PBase_j$ based on the utilization of the resource on compute node j from time AT_i to DT_i, where AT_i is the time when job i arrives at the cluster and DT_i is the deadline time which job i has to be completed:

$$PUtil_{ij} = \frac{RESMax_j}{RESFree_{ij}} * PBase_j \tag{4}$$

$RESMax_j$ is the maximum units of the resource on compute node j from time AT_i to DT_i. $RESFree_{ij}$ is the remaining free units of the resource on compute node j from time AT_i to DT_i, after deducting units of resource committed for other current executing jobs and job i from the maximum units of the resource:

$$RESFree_{ij} = RESMax_j - \left(\sum_{k=1}^{n_{accept_j}} RES_k \right) - RES_i \quad (5)$$

For n jobs submitted to the cluster, n_{accept} jobs are accepted for execution by the admission control. If there is no admission control, $n_{accept} = n$. We define n_{accept_j} to be n_{accept} jobs that are executing on compute node j from time AT_i to DT_i. Our Economy-based Admission Control and Resource Allocation mechanisms first check that there is sufficient resource on node j before computing its pricing rate P_{ij} so that $RESFree_{ij}$ is always positive.

The pricing function computes the pricing rate P_{ij} for each different resource to be used by job i on compute node j. Thus, the overall pricing rate of executing job i on compute node j can be computed as the sum of each P_{ij}. This fine-grain pricing is fair since jobs are priced based on the amount of different resources utilized. For instance, a compute-intensive job does not require a large disk storage size as compared to a data-intensive job and therefore is priced significantly lower for using the disk storage resource.

The pricing function provides incentives that takes into account both user-centric and system-centric factors. The user-centric factor considered is the amount of a resource RES_i required by job i. For example, a low amount of the required resource (a low RES_i) results in a low pricing rate P_{ij}. The system-centric factor taken into account is the availability of the required resource $RESFree_{ij}$ on the compute node j. For instance, the required resource that is low in demand on node j (a high $RESFree_{ij}$) will have a low pricing rate P_{ij}.

Libra [11] gives incentives to jobs with long deadlines as compared to jobs with short deadlines irrespective of the cluster workload condition. Instead, our proposed pricing function considers the cluster workload because the utilization pricing rate $PUtil_{ij}$ considers the utilization of a resource based on the required deadline of job i (from time AT_i to DT_i). Consider this example where the user specifies a short deadline and long deadline of 2 and 5 hours respectively to execute a job i that requires 10 units of memory size. For the compute node j, we assume that the base pricing rate $PBase_j$ is \$1 per unit, there are 100 free units of memory size for each hour of deadline, and there are n_{accept_j} jobs using 90 units of memory size during both deadlines. So, for a short deadline of 2 hours, $PUtil_{ij} = (200/(200 - 90 - 10)) * 1 = \$2$ per unit. Whereas, for a long deadline of 5 hours, $PUtil_{ij} = (500/(500 - 90 - 10)) * 1 = \$1.25$ per unit which is lower.

Our proposed pricing function is dynamic since the overall pricing rate of a job varies depending on the availability of each resource on different compute nodes for the required deadline. It is also adaptive as the overall pricing rate is adjusted automatically depending on the current supply and demand of resources to either encourage or discourage job submission.

VI. Mechanisms for Enforcing Required Utility

We enhance the admission control and resource allocation mechanisms from Libra [11] to incorporate the proposed user-level job submission specification that solicits fine-grain user-centric information for jobs and the proposed pricing function that computes dynamic and adaptive pricing for resources.

A. Economy-based Admission Control and Resource Allocation

We consider utility-driven resource management and allocation in a simplified cluster operating environment with the following assumptions:

1) The users submit parallel compute-intensive jobs that require at least one or more processors into the cluster for execution.
2) The runtime estimate of each job is known and given during job submission and is correct. We envision that the nature of the jobs submitted enables their runtimes to be predicted in advance based on means such as past execution history.
3) The QoS requirements specified by the user during job submission do not change after the job is accepted.
4) The cluster RMS is the only gateway for users to submit jobs to the cluster. In other words, all compute nodes in the cluster are dedicated for executing jobs that can only be assigned by the cluster RMS. This also implies that the cluster RMS has the full knowledge of allocated workload currently in execution and the resources available on each compute node.
5) The compute nodes can be homogeneous (have the same hardware architectures) or heterogeneous (have different hardware architectures). For heterogeneous compute nodes, the runtime estimate is translated to its equivalent on the allocated compute node.
6) The underlying operating system at the compute nodes supports time-shared execution mechanism. A time-shared execution mechanism allows multiple jobs to be executed on a compute node at the same time. Each job shares processor time by executing within assigned processor time partitions.

Currently, our enhanced Economy-based Admission Control and Resource Allocation mechanisms use a simplified version of the job submission specification in (2) that excludes the $QoSOptimization$ segment for the parallel compute-intensive jobs:

1) $JobDetails$:
   a) Runtime estimate RE_i
2) $ResourceRequirements$:
   a) Memory size MEM_i
   b) Disk storage size $DISK_i$
   c) Number of processors $PROC_i$
3) $QoSConstraints$:
   a) Deadline D_i
   b) Budget B_i

In addition, it only considers a single cluster resource which is the processor time utilized by the job. In this case, the proposed pricing function only computes the pricing rate for the processor time resource. So, $RESFree_{ij}$ which is the free processor time resource on compute node j from time AT_i to DT_i, excluding the runtime estimate RE_k used by other current executing jobs and RE_i by job i is defined as:

$$RESFree_{ij} = RESMax_j - \left(\sum_{k=1}^{n_{accept_j}} RE_k \right) - RE_i \tag{6}$$

Our enhanced Economy-based Admission Control and Resource Allocation mechanisms determine whether a job can be accepted or rejected based on three criteria:

1) Resources required by the job to be executed
2) Deadline that the job has to be finished
3) Budget to be paid by the user for the job to be finished within the deadline

Algorithm 1 shows the pseudocode for the enhanced Economy-based Admission Control and Resource Allocation mechanisms using the proposed pricing function. First, the Admission Control mechanism determines whether there is sufficient number of compute nodes that can fulfill the specified resource requirements for job i (line 1–8). This rejects jobs that require more resources than that can be supported by the cluster. Then, the Admission Control mechanism determines whether there is sufficient number of these compute nodes that can fulfill the required deadline time DT_i and has the required resources for job i with runtime estimate RE_i (line 9–16). These compute nodes are then sorted in ascending order using $RESFree_{ij}$ in (6) (line 17). This ensures that each compute node is allocated jobs to its maximum capacity so that more jobs can be accepted and completed within their required deadlines. The first $PROC_i$ number of compute nodes in the sorted list that is within the specified budget B_i is then allocated to job i (line 18–34). A node j is allocated if its cost for utilizing the processor time resource based on the pricing rate P_{ij} in (3) is not more than the specified budget B_i of job i (line 19–25). The cost C_i for job i is thus computed as the highest cost needed from the allocated $PROC_i$ number of allocated compute nodes (line 22–24). If there is sufficient number of compute nodes meeting B_i, job i is accepted, else it is rejected (line 30–34).

B. Job Control

The Job Control mechanism at each compute node needs to enforce the QoS of a job so as to ensure that the job can finish with the required utility. Currently, we only consider enforcing a single QoS which is the deadline. We adopt the time-shared job control mechanism from Libra [11] that implements proportional-share resource allocation based on the required deadline of the job. The Job Control mechanism computes the initial processor time partition for a newly started job and then periodically updates processor time partitions for all current executing jobs to enforce that their deadlines can be satisfied.

Algorithm 2 shows the pseudocode for the Job Control mechanism that computes the processor time partition for each job i that is executing on a compute node j. The job control updates new processor time partition for every executing job i based on the processor clock time completed so far and the actual wall clock time taken with respect to its runtime estimate RE_i and deadline D_i (line 1–4).

VII. Performance Evaluation

We evaluate the performance of our enhanced deadline-based proportional processor share policy (referred to as Libra+$) through simulation for varying cluster workload, varying pricing factor and tolerance against runtime under-estimates.

Algorithm 1: Pseudocode for Economy-based Admission Control and Resource Allocation mechanisms.

for $j \leftarrow 0$ ***to*** c **do**
  if *node j has required resources* **then**
    place node j in $ListResReq_i$;
  endif
endfor
if $ListResReq_i\_size < PROC_i$ **then**
  reject job i with cannot_meet_resources message;
else
  for $j \leftarrow 0$ ***to*** $ListResReq_i\_size - 1$ **do**
    if *node j can finish job i with RE_i before DT_i* **and** *node j has required resources for RE_i* **then**
      place node j in $ListDeadline_i$;
    endif
  endfor
  if $ListDeadline_i\_size < PROC_i$ **then**
    reject job i with cannot_meet_deadline message;
  else
    sort $ListDeadline_i$ by $RESFree_{ij}$ in ascending order;
    for $j \leftarrow 0$ ***to*** $ListDeadline_i\_size - 1$ **do**
      compute P_{ij};
      if $RE_i * P_{ij} \leq B_i$ **then**
        place node j in $ListAllocation_i$;
        if $ListAllocation_i\_size = 1$ *or* $RE_i * P_{ij} > C_i$ **then**
          $C_i = RE_i * P_{ij}$;
        endif
      endif
      if $ListAllocation_i\_size = PROC_i$ **then**
        break;
      endif
    endfor
    if $ListAllocation_i\_size < PROC_i$ **then**
      reject job i with cannot_meet_budget message;
    else
      accept job i and allocate job i to all nodes in $ListAllocation_i$;
    endif
  endif
endif

Algorithm 2: Pseudocode for Job Control mechanism.

for *all job i executing on compute node j* **do**
  get processor clock time $clockCPU_{ij}$ completed so far by node j for job i;
  get wall clock time $clockWall_i$ currently taken by job i;
  set processor time partition $Partition_{ij} = \frac{RE_i - clockCPU_{ij}}{D_i - clockWall_i}$;
endfor

A. Evaluation Metrics

We define two user-centric performance evaluation metrics to measure the level of utility achieved by the cluster: Job QoS Satisfaction and Cluster Profitability.

Job QoS Satisfaction measures the level of utility for satisfying job requests. A higher Job QoS Satisfaction represents better performance. It is computed as the proportion of n_{QoS} jobs whose required QoS (deadline and budget) are fulfilled out of n jobs submitted:

$$\textit{Job QoS Satisfaction} = n_{QoS}/n \tag{7}$$

n_{QoS} is n_{accept} jobs (accepted by the admission control) with their required QoS satisfied. Currently, we only consider two basic QoS parameters: deadline D_i and budget B_i. To satisfy D_i, the finish time must be less than the deadline time, that is $FT_i \leq DT_i$. To satisfy B_i, the actual cost paid by the user must be less than the budget, that is $C_i \leq B_i$.

Cluster Profitability measures the level of utility for generating economic benefits for the cluster owner. A higher Cluster Profitability denotes better performance. It is computed as the proportion of profit earned by the cluster for meeting job QoS out of the total budget of jobs that are submitted:

$$\textit{Cluster Profitability} = \sum_{i=1}^{n_{QoS}} C_i / \sum_{i=1}^{n} B_i \tag{8}$$

TABLE I

DEFAULT SIMULATION SETTINGS.

Parameter	Default value
% of high urgency jobs	20.0
% of low urgency jobs	80.0
Deadline high:low ratio	4.0
Deadline low mean	2.0
Budget high:low ratio	4.0
Budget low mean	2.0
Arrival delay factor	0.5
% of inaccuracy	0.0
Libra+$:	
Static pricing factor α	1.0
Dynamic pricing factor β	0.1

B. Experimental Methodology

We use GridSim [18] to evaluate the performance. GridSim provides an underlying infrastructure that allows construction of simulation models for heterogeneous resources, users, applications and schedulers. GridSim has been used for the design and evaluation of economy-based scheduling algorithms in both cluster [11] and Grid computing [19][20].

For the experiments, we use a subset of the last 5000 jobs in the SDSC SP2 trace (April 1998 to April 2000) version 2.2 from Feitelson's Parallel Workload Archive [23]. This 5000 job subset is based on the last 3.75 months of the SDSC SP2 trace and requires an average of 17 processors, average inter arrival time of 1969 seconds (32.8 minutes), and average runtime of 8671 seconds (2.4 hours).

As QoS parameters (deadline and budget) are not recorded in the trace, we follow a similar experimental methodology in [24] to model these parameters through two job classes: (i) high urgency and (ii) low urgency. Each job in the *high urgency* class has a deadline of low D_i/R_i value and budget of high $B_i/f(R_i)$ value. $f(R_i)$ is a function to represent the minimum budget the user will quote with respect to R_i. Conversely, each job in the *low urgency* class has a deadline of high D_i/R_i value and budget of low $B_i/f(R_i)$ value. This model is realistic since a user who submits a more urgent job to be completed within a shorter deadline is likely to offer a higher budget for the job to be finished on time. The arrival sequence of jobs from the high urgency and low urgency classes is randomly distributed.

Values are normally distributed within each of deadline and budget parameters. The ratio of the means for each parameter's high-value and low-value is thus known as the *high:low ratio*. So, a higher budget high:low ratio denotes that high urgency jobs have larger budgets that of a lower ratio. For instance, a budget high:low ratio of 8 means the B_i/R_i mean of high urgency jobs is two times more than that of a budget high:low ratio of 4.

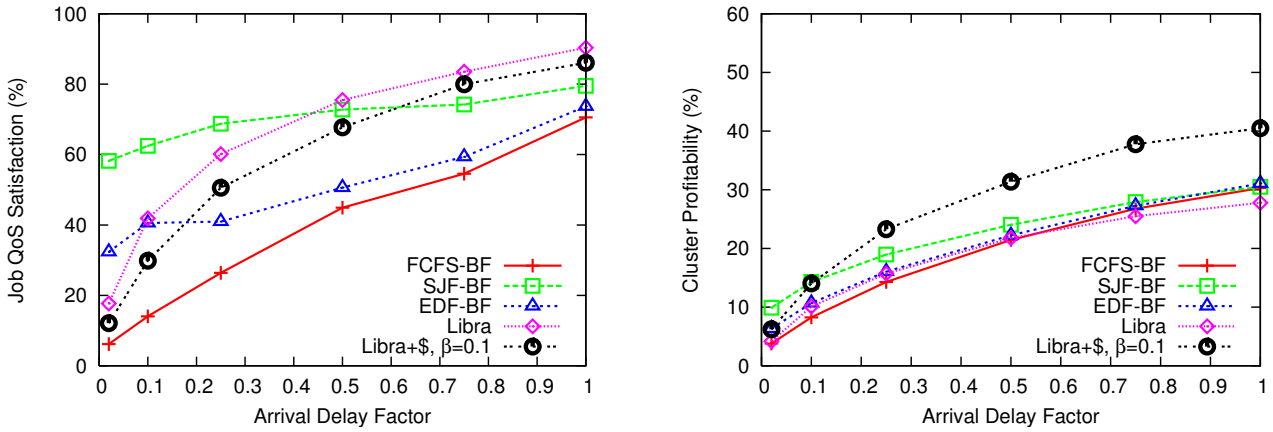
Table I lists the default settings for our simulations. We model varying workload thru the *arrival delay factor* which sets the arrival delay of jobs based on the inter arrival time from the trace. For example, an arrival delay factor of 0.1 means a job with 600 seconds of inter arrival time from the trace now has a simulated inter arrival time of 60 seconds. Hence, a lower delay factor represents higher workload by shortening the inter arrival time of jobs.

We investigate the tolerance against runtime under-estimates so as to examine the impact of delays caused by earlier jobs on later jobs, in particular failing to meet their required QoS. We do not consider over-estimated value of RE_i since jobs accepted by the admission control will still be completed within their required deadlines. The *inaccuracy* of runtime under-estimates is computed with respect to the real runtime from the trace. An inaccuracy of 0% assumes runtime estimates are equal to the real runtimes of the jobs, while a higher percentage denotes higher under-estimation.

For the operating environment, we simulate the 128-node IBM SP2 located at San Diego Supercomputer Center (SDSC) (where the selected trace originates from) with the following characteristics:

- SPEC rating of each compute node: 168
- Number of compute nodes: 128
- Processor type on each compute node: RISC System/6000
- Operating System: AIX

To facilitate comparison, we also model three backfilling resource allocation policies: (i) FCFS-BF, (ii) SJF-BF, and (iii) EDF-BF which prioritize jobs based on arrival time (First Come First Serve), runtime estimate (Shortest Job First), and deadline (Earliest Deadline First) respectively, in addition to Libra [11] and Libra+$. These three policies are implemented as variations of EASY backfilling [21][22] that assigns unused processors to the next waiting jobs in the queue based on their runtime estimates, provided that they do not delay the first job with highest priority. This means that each job needs to wait for its



(a) Job QoS Satisfaction

(b) Cluster Profitability

Fig. 2. Impact of decreasing workload. Libra+$ generates higher Cluster Profitability than all other policies even though it manages similar or lower Job QoS Satisfaction, thus demonstrating the effectiveness of its pricing function in improving the economic benefit of the cluster owner.

required number of processors to be available in order to run since only a single job can run on a processor at any time (i.e. space-shared).

We select EASY backfilling for comparison since it is currently supported and widely used in a number of schedulers, including LoadLeveler [13], LSF [14], and Maui [17]. However, these backfilling policies have very poor performance without job admission control as they are not able to restrict the number of jobs which can overload the cluster. So, we have modified these policies not to run any waiting jobs that have exceeded their deadlines from the queue; the removed jobs are simply treated as rejected. This generous admission control not only allows the policies to choose their highest priority job at the latest time, but also ensures that jobs whose deadlines have lapsed are not unnecessarily run to incur propagated delay for later jobs.

In order to measure the Cluster Profitability metric, we also model the three backfilling policies to incorporate a simple pricing mechanism. The profit of processing a job is only considered when the deadline of the job is met. The user is then charged based on the static base pricing rate $PBase_j$ of using processing time on node j, so job i has its cost $C_i = RE_i * PBase_j$.

Libra [11] uses a pricing function that provides incentives for jobs with more relaxed deadlines to compute a static cost, so job i has its cost $C_i = \gamma * RE_i + \delta * RE_i/D_i$. γ is a factor for the first component that computes the cost based on the runtime of the job, so that longer jobs are charged more than shorter jobs. δ is a factor for the second component that provides incentives for jobs with more relaxed deadlines, so as to encourage users to submit jobs with longer deadlines. For our experiments, $\gamma = 1$ and $\delta = 1$. Libra is used to evaluate the effectiveness of the proposed pricing function in Libra+$ for improving utility for the cluster owner.

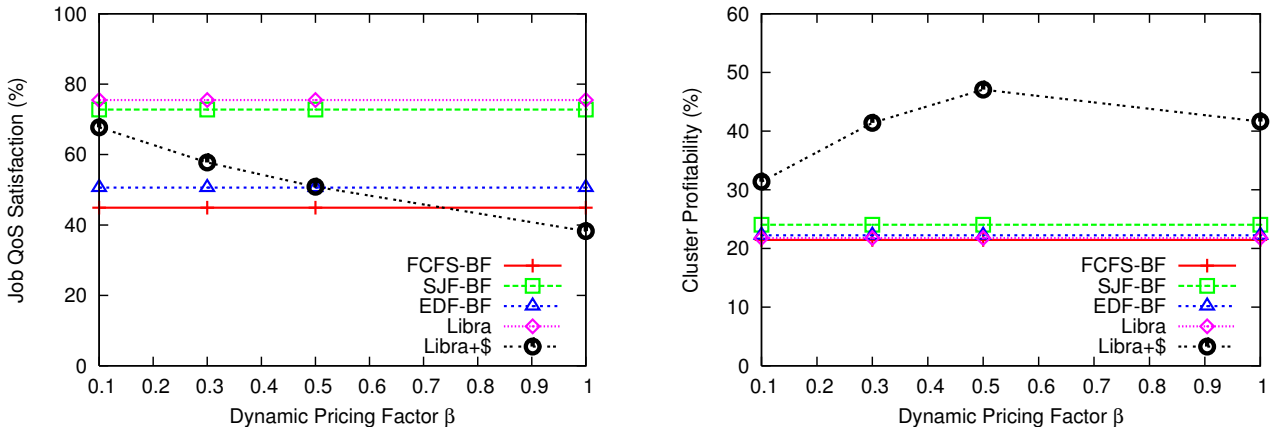
C. Varying Cluster Workload

Figure 2 shows how Libra+$ performs with respect to other policies for changing cluster workload. We can see that a decreasing workload (increasing arrival delay factor) results in an increasing Job QoS Satisfaction and Cluster Profitability as more jobs can have their deadlines met.

For Job QoS Satisfaction (Figure 2(a)), both Libra and Libra+$ are able to achieve substantially higher performance than FCFS-BF and EDF-BF since they consider the required QoS (deadline and budget) of each different job and allocate resources proportionally to each job based on its required deadline so that more jobs can be satisfied. However, Libra+$ has a lower Job QoS satisfaction as compared to Libra. This is because the proposed pricing function computes higher pricing, thus denying jobs with insufficient budgets.

When the cluster workload is high (with lower arrival delay factor), both Libra and Libra+$ have a lower Job QoS Satisfaction than SJF-BF and EDF-BF. This is due to the fact that in our experiments, the deadlines are always set as larger factors of runtimes. Thus, SJF-BF and EDF-BF are able to exploit this as they have a more favorable selection choice due to their generous admission controls that we implemented to reject jobs only when their deadlines have lapsed, and not when they are submitted. However, both Libra and Libra+$ are able to overcome this unfairness, as seen in achieving higher Job QoS Satisfaction than SJF-BF and EDF-BF as workload decreases.

On the contrary, Figure 2(b) shows that the proposed pricing function enables Libra+$ to continue generating significantly higher Cluster Profitability than Libra and SJF-BF even though fewer jobs are accepted, thus proving its effectiveness in improving the cluster owner's economic benefits. Accepting fewer but higher-priced jobs enables Libra+$ to maintain a higher Cluster Profitability to compensate for a lower Job QoS Satisfaction.



(a) Job QoS Satisfaction

(b) Cluster Profitability

Fig. 3. Impact of increasing dynamic pricing factor β. A higher β returns lower Job QoS Satisfaction, but higher Cluster Profitability. The cluster owner can adjust the value of β to determine the level of sharing for the cluster.

Another interesting point to note is that EDF-BF returns a similar Cluster Profitability as FCFS-BF even though it has a higher Job QoS Satisfaction. This may be due to more urgent jobs being completed first in EDF-BF, but in turn resulting in other less urgent jobs exceeding their deadlines, thus returning similar Cluster Profitability as FCFS-BF.

D. Varying Pricing Factor for Different Level of Sharing

We study the level of sharing supported by Libra+$. We increase the dynamic pricing factor β to observe its impact on Libra+$ in supporting the level of sharing.

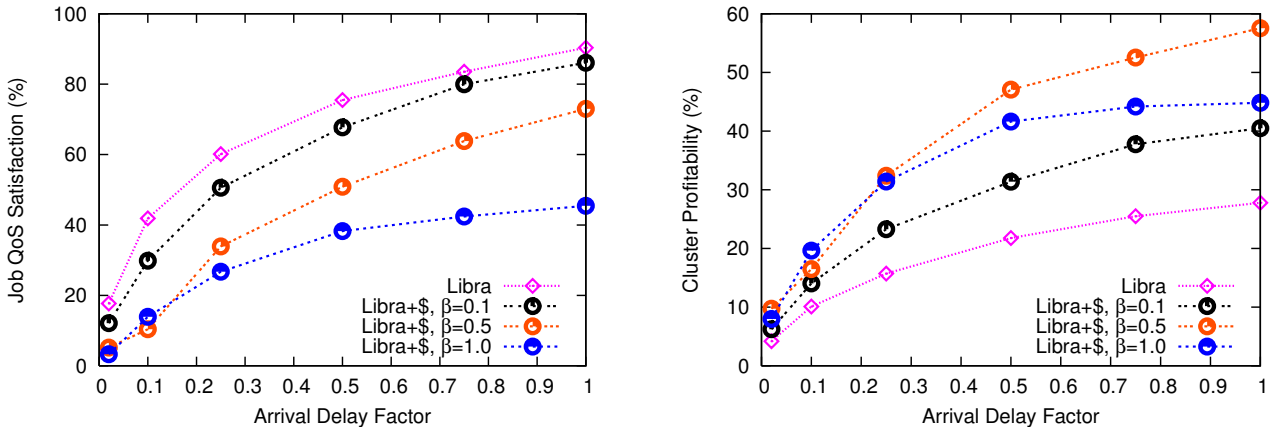
Figure 3(a) shows that an increasing β for Libra+$ results in decreasing Job QoS Satisfaction, while Figure 3(b) shows that it results in increasing Cluster Profitability. This demonstrates that the proposed pricing function is able to generate increasing profit even though a decreasing number of jobs is accepted. This is possible since jobs with sufficient budgets are executed at a higher cost (due to higher β) to compensate for accepting fewer jobs due to insufficient budgets. A point to note from Fig. 3(a) is that if β is set too high, the Job QoS Satisfaction can be lower than other policies such as EDF-BF and FCFS-BF. An increasing β will also ultimately result in lower Cluster Profitability (eg. $\beta = 1.0$ in Figure 3(b)) since too many jobs are rejected by the high quoted prices.

These results show that the dynamic pricing factor β has a significant impact on both Job QoS Satisfaction and Cluster Profitability. A higher β lowers the level of sharing (a lower Job QoS Satisfaction), but increases the economic benefit of the cluster owner (a higher Cluster Profitability). However, β should not be set too high such that more jobs will be rejected and result in lower profit. Thus, the cluster owner can determine the level of sharing by adjusting the value of β. This demonstrates the flexibility of the pricing function in enabling the cluster owner to easily configure and determine the level of sharing based on his objective.

Figure 4 presents the impact of decreasing workload on the dynamic pricing factor β. Figure 4(b) shows that Libra+$ always return a higher Cluster Profitability than Libra, thus demonstrating the effectiveness of its pricing function in computing pricing based on demand (higher pricing for higher workload). This can also be verified by observing Libra+$ achieving higher Cluster Profitability when the workload is high (arrival delay factor $<= 0.5$) in Figure 4(b), though it manages lower Job QoS Satisfaction in Figure 4(a). In particular, a higher β is still able to generate higher Cluster Profitability with lower Job QoS Satisfaction.

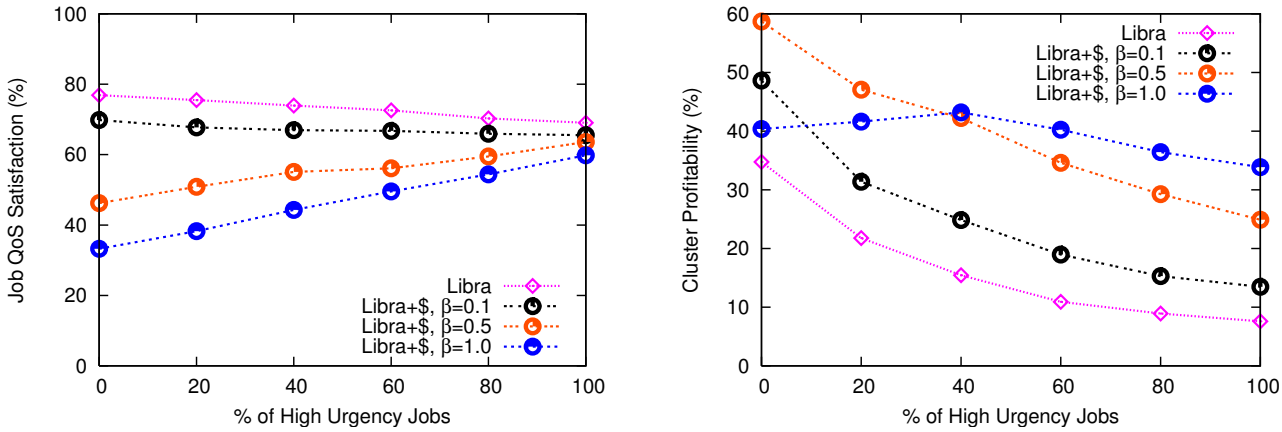
A larger increase in Cluster Profitability is also obtained for higher β as the workload decreases. For example, the Cluster Profitability in Figure 4(b) increases by 25% for $\beta = 0.5$ (from 32% when the arrival delay factor is 0.25 to 57% when the arrival delay factor is 1.0). On the other hand, there is only an increase of 17% in Cluster Profitability for $\beta = 0.1$ (from 23% when the arrival delay factor is 0.25 to 40% when the arrival delay factor is 1.0). But again, if β is set too high (eg. $\beta = 1.0$ compared to $\beta = 0.5$), a smaller increase in Cluster Profitability is obtained instead (only 13% increase from 31% when the arrival delay factor is 0.25 to 44% when the arrival delay factor is 1.0).

However, Figure 5(b) shows that a high β (eg. $\beta = 1.0$) is able to generate higher Cluster Profitability when there are more high urgency jobs, but it can result in lower Cluster Profitability when there are less high urgency jobs. In addition, as seen in Figure 5(a), a high β also has higher Job QoS Satisfaction that gradually increases to almost equivalent to that of low β. This reinforces the need for a cluster owner to be aware of how the configuration of the dynamic pricing factor can easily influence the level of sharing and profit earned for his cluster.



(a) Job QoS Satisfaction (b) Cluster Profitability

Fig. 4. Impact of decreasing workload with dynamic pricing factor β. Decreasing workload results in a larger increase in Cluster Profitability for higher β, but smaller increase if β is set too high.



(a) Job QoS Satisfaction (b) Cluster Profitability

Fig. 5. Impact of increasing number of high urgency jobs with dynamic pricing factor β. A higher β will generate higher Cluster Profitability when there are more high urgency jobs, even though it can result in lower Cluster Profitability when there are less high urgency jobs.

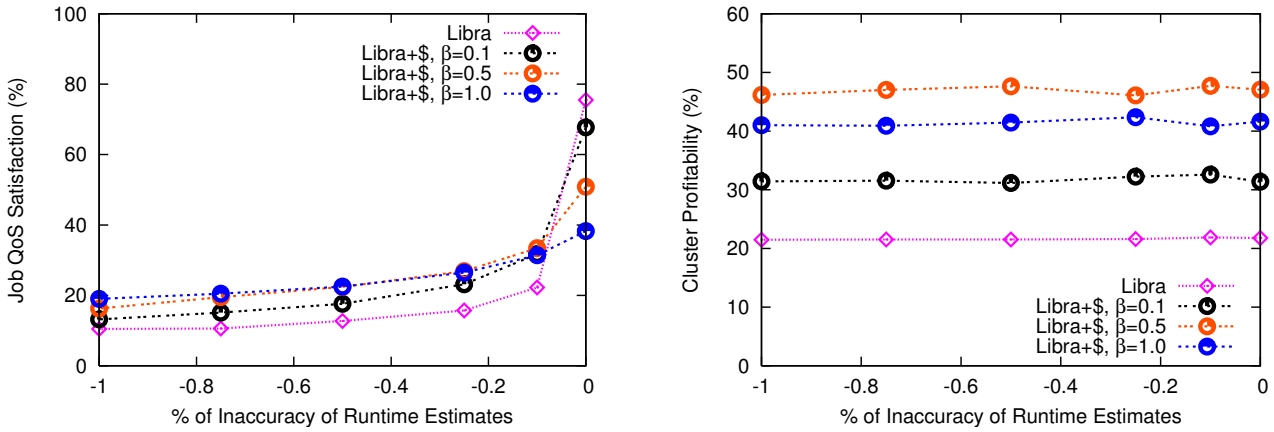
E. Tolerance against runtime under-estimates

Figure 6(a) shows that when there is no runtime under-estimates (0% of inaccuracy of runtime estimates), a higher dynamic pricing factor β for Libra+$ results in much lower Job QoS Satisfactions. But, with increasing under-estimates (increasing negative inaccuracy), a higher β results in higher Job QoS satisfaction, whereas Libra has the lowest Job QoS satisfaction. This shows that a higher β provides a higher degree of tolerance against runtime under-estimates since fewer jobs are accepted and thus the possibility of delays occurring on later jobs is lower.

Figure 6(b) shows that increasing runtime under-estimates results in the lowest Cluster Profitability for Libra as fewer jobs are accepted due to delays caused by earlier jobs. However, a higher $\beta =$ for Libra+$ can still maintain higher Cluster Profitability with increasing runtime under-estimates. This reiterates the effectiveness of the proposed pricing function in improving the economic benefit of the cluster owner, even in the case of runtime under-estimates.

VIII. Conclusion

We have demonstrated the importance of an effective pricing mechanism for achieving utility-driven resource management and allocation in clusters, especially when demand exceeds supply of cluster resources. We show that our enhanced pricing function meets the four essential requirements for pricing of resources. In particular, our pricing function provides flexibility for the cluster owner to easily configure the pricing of his cluster resources to modify the level of sharing. Our pricing function also adapts to the changing supply and demand of resources by computing higher pricing when cluster workload increases. This serves as an effective means for admission control to prevent the cluster from overloading and tolerate against job runtime under-estimates. Finally, the pricing function generates a higher economic benefit for the cluster owner.



(a) Job QoS Satisfaction

(b) Cluster Profitability

Fig. 6. Impact of increasing runtime under-estimates with dynamic pricing factor β. A higher β provides a higher level of tolerance against runtime under-estimates (better performance) for both Job QoS Satisfaction and Cluster Profitability.

ACKNOWLEDGMENT

We thank Anthony Sulistio for his support with the use of GridSim.

REFERENCES

[1] G. F. Pfister, *In Search of Clusters*, 2nd ed. Prentice Hall PTR, 1998.
[2] R. Buyya, Ed., *High Performance Cluster Computing: Architectures and Systems*. Prentice Hall PTR, 1999.
[3] R. Buyya, T. Cortes, and H. Jin, "Single System Image," *The International Journal of High Performance Computing Applications*, vol. 15, no. 2, pp. 124–135, Summer 2001.
[4] R. Buyya, D. Abramson, and J. Giddy, "A Case for Economy Grid Architecture for Service Oriented Grid Computing," in *Proc. of 10th IEEE International Heterogeneous Computing Workshop (HCW 2001)*, San Francisco, CA, Apr. 2001.
[5] B. N. Chun and D. E. Culler, "User-centric Performance Analysis of Market-based Cluster Batch Schedulers," in *Proc. of 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, Berlin, Germany, May 2002.
[6] R. Buyya, D. Abramson, and J. Giddy, "An Economy Driven Resource Management Architecture for Global Computational Power Grids," in *Proc. of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, Las Vegas, NV, June 2000.
[7] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta, "Spawn: A Distributed Computational Economy," *IEEE Trans. Software Eng.*, vol. 18, no. 2, pp. 103–117, Feb. 1992.
[8] 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." in *Proc. of 3rd International Conference on Parallel and Distributed Information Systems (PDIS '94)*, Austin, TX, Sept. 1994.
[9] B. N. Chun and D. E. Culler, "Market-based Proportional Resource Sharing for Clusters," University of California at Berkeley, Computer Science Division, Technical Report CSD-1092, Jan. 2000.
[10] B. N. Chun and D. E. Culler, "REXEC: A Decentralized, Secure Remote Execution Environment for Clusters," in *Proc. of 4th Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing (CANPC '00)*, Toulouse, France, Jan. 2000.
[11] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya, "Libra: a computational economy-based job scheduling system for clusters," *Software: Practice and Experience*, vol. 34, no. 6, pp. 573–590, May 2004.
[12] *Condor Version 6.7.1 Manual*, University of Wisconsin-Madison, 2004. [Online]. Available: http://www.cs.wisc.edu/condor/manual/v6.7
[13] *LoadLeveler for AIX 5L Version 3.2 Using and Administering*, SA22-7881-01, IBM, Oct. 2003.
[14] *LSF Version 4.1 Administrator's Guide*, Platform Computing, 2001.
[15] *OpenPBS Release 2.3 Administrator Guide*, Altair Grid Technologies, Aug. 2000.
[16] *Sun ONE Grid Engine, Administration and User's Guide*, Sun Microsystems, Oct. 2002.
[17] *Maui Scheduler Version 3.2 Administrator's Guide*, Supercluster Research and Development Group, 2004. [Online]. Available: http://www.supercluster.org/mauidocs/mauiadmin.shtml
[18] A. Sulistio, G. Poduvaly, R. Buyya, and C.-K. Tham, "Constructing A Grid Simulation with Differentiated Network Service Using GridSim," in *Proc. of 6th International Conference on Internet Computing (ICOMP 2005)*, Las Vegas, NV, June 2005.
[19] R. Buyya, J. Giddy, and D. Abramson, "An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications," in *Proc. of 2nd Annual Workshop on Active Middleware Services (AMS 2000)*, Pittsburgh, PA, Aug. 2000.
[20] R. Buyya, M. Murshed, and D. Abramson, "A Deadline and Budget Constrained Cost-Time Optimization Algorithm for Scheduling Task Farming Applications on Global Grids," in *Proc. of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2002)*, Las Vegas, NV, June 2002.
[21] D. Lifka, "The ANL/IBM SP Scheduling System," in *Proc. of 1st Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 1995)*, Santa Barbara, CA, Apr. 1995.
[22] A. W. Mu'alem and D. G. Feitelson, "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling," *IEEE Trans. Parallel and Distributed Systems*, vol. 12, no. 6, pp. 529–543, June 2001.
[23] (2006, Oct.) Parallel Workloads Archive. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload
[24] D. E. Irwin, L. E. Grit, and J. S. Chase, "Balancing Risk and Reward in a Market-based Task Service," in *Proc. of 13th International Symposium of High Performance Distributed Computing (HPDC13)*, Honolulu, HI, June 2004.