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}@cs.mu.oz.au

***Abstract*—Users perceive varying level 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 user to express their 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 *computation 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 uses 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 computation-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 valuation 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 user 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 computation 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 computation 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

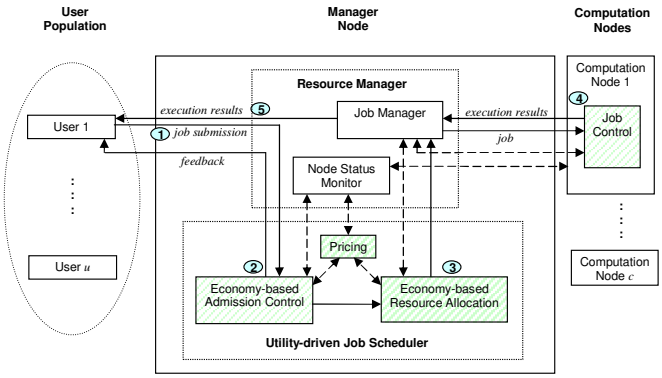


Fig. 1. Architectural framework for a utility-driven cluster RMS. The Economy-based Admission Control mechanism determines whether a job submitted into the cluster should be accepted or rejected and feedback to the user. If accepted, the Economy-based Resource Allocation mechanism determines the best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

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.

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 computation 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 computation 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 computation node the job is to be allocated to. The job manager is then informed to dispatch the job to the selected computation 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:

$$job_i(\ [Segment_1]\ [Segment_2]\ ...\ [Segment_s]\ ) \quad (1)$$

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 sequential compute-intensive single-task job:

$$job_i(\ [JobDetails]\ [ResourceRequirements]\ [QoSConstraints]\ [QoSOptimization]) \quad (2)$$

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) Estimated execution time EE_i: the estimated execution time for job i on a computation node. We define the execution time E_i of job i as the time period for it to be processed on a computation node provided that it is allocated the node's full proportion of processing power. Thus, the execution time varies on nodes of different hardware and software architecture and does not include any waiting time and communication latency. The execution time 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 computation node. This facilitates the cluster RMS to determine whether a computation 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.

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 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 computation 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 sequential compute-intensive single-task 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 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 computation node j as:

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

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 computation 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 computation 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:

$$PUtil_{ij} = \frac{RESMax_j}{RESFree_{ij}} * PBase_j \quad (4)$$

$RESMax_j$ is the maximum units of the resource on computation node j from time AT_i to DT_i. $RESFree_{ij}$ is the remaining free units of the resource on computation 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 computation 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 computation node j. Thus, the overall pricing rate of executing job i on computation

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 computation 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 computation 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 computation 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 only sequential compute-intensive single-task jobs into the cluster for execution.
2) The estimated execution time of each job is known and given during job submission and is correct. We envision that the nature of the jobs submitted enables their execution time 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 computation 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 computation node.
5) The computation nodes can be homogeneous (have the same hardware architectures) or heterogeneous (have different hardware architectures). For heterogeneous computation nodes, the estimated execution time is translated to its equivalent on the allocated computation node.
6) The underlying operating system at the computation nodes supports time-shared execution mechanism. A time-shared execution mechanism allows multiple jobs to be executed on a computation 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 sequential compute-intensive single-task jobs:

1) *JobDetails*:
   a. Estimated execution time EE_i
2) *ResourceRequirements*:
   a. Memory size MEM_i
   b. Disk storage size $DISK_i$
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 computation node j from time AT_i to DT_i, excluding the estimated execution time EE_k used by other current executing jobs and EE_i by job i is defined as:

$$RESFree_{ij} = RESMax_j - \left(\sum_{k=1}^{n_{accept_j}} EE_k \right) - EE_i \quad (6)$$

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

Algorithm 1. Pseudo-code for Economy-based Admission Control and Resource Allocation mechanisms.

```
1  for j ← 0 to c do
2     if node j has required resources then
3        place node j in ListResReq_i ;
4     endif
5  endfor
6  if ListResReq_i is empty then
7     reject job i with cannot_meet_resources message;
8  else
9     for j ← 0 to ListResReq_i_size – 1 do
10       if node j can finish job i with EE_i before DT_i and
         node j has required resources for EE_i then
11          compute P_ij ;
12          place node j in ListDeadline_i ;
13       endif
14    endfor
15    if ListDeadline_i is empty then
16       reject job i with cannot_meet_deadline message;
17    else
18       sort ListDeadline_i by RESFree_ij in ascending order;
19       for j ← 0 to ListDeadline_i_size – 1 do
20          if EE_i * P_ij ≤ B_i then
21             accept and allocate job i to node j;
22             break;
23          endif
24       endfor
25       reject job i with cannot_meet_budget message;
26    endif
27 endif
```

Algorithm 2. Pseudo-code for Job Control mechanism.

```
1  for all job i executing on computation node j do
2     get processor clock time clockCPU_ij completed so far by node j
      for job i;
3     get wall clock time clockWall_i currently taken by job i;
4     set processor time partition Partition_ij = (EE_i – clockCPU_ij) / (D_i – clockWall_i) ;
5  endfor
```

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 pesudo-code for the enhanced Economy-based Admission Control and Resource Allocation mechanisms using the proposed pricing function. First, the Admission Control mechanism determines whether there are any computation nodes that can fulfill the specified resource requirements for job i (line 1–7). This rejects jobs that require more resources than that can be supported by the cluster. Then, the Admission Control mechanism determines whether there are any of these computation nodes that can fulfill the required deadline time DT_i and has the required resources for job i with estimated execution time EE_i (line 9–16). Each computation node j that has the required resources and can fulfill the required deadline time DT_i also computes the pricing rate P_{ij} for utilizing the processor time resource (line 11). These computation nodes are then sorted in ascending order using $RESFree_{ij}$ in (6) (line 18). The first computation node j in the sorted list that is within the specified budget B_i is allocated the job i (line 19–25). This ensures that each computation node is allocated jobs to their maximum capacity so that more jobs can be accepted and completed within their required deadline.

B. Job Control

The Job Control mechanism at each computation 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 deadline can be satisfied.

Algorithm 2 shows the pesudo-code for the Job Control mechanism that computes the processor time partition for each job i that is executing on a computation node j. The job control updates new processor time partition for every executing jobs based on the processor clock time completed so far and the actual wall clock time taken with respect to their estimated execution time EE_i and deadline D_i (line 1–4).

VII. Performance Evaluation

We simulate a cluster and carry out detailed evaluation using both user-centric and system-centric evaluation metrics. We evaluate performance of our proposed Economy-based Admission Control and Resource Allocation with respect to varying cluster workload, varying pricing factor and tolerance against estimation error for estimated execution time.

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:

$$Job\ QoS\ Satisfaction = n_{QoS} / n \quad (7)$$

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 $FT_i \le DT_i$. To satisfy B_i, the actual cost paid by the user must be less than the budget, that is $C_i \le B_i$.

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 out of the total budget of jobs that are accepted for execution:

$$Cluster\ Profitability = \sum_{i=1}^{n_{accept}} C_i \Big/ \sum_{i=1}^{n_{accept}} B_i \quad (8)$$

We also use two traditional system-centric performance evaluation metrics: Average Waiting Time and Average Response Time.

Average Waiting Time is the average time a job waits in the cluster before it starts execution. A lower Average Waiting Time indicates better performance.

$$Average\ Waiting\ Time = \frac{1}{n_{accept}} \sum_{i=1}^{n_{accept}} ST_i - AT_i \quad (9)$$

ST_i is the time when job i starts execution on a computation node.

Average Response Time is the average time a job is completed and results returned to the user. A lower Average Response Time signifies better performance.

$$Average\ Response\ Time = \frac{1}{n_{accept}} \sum_{i=1}^{n_{accept}} FT_i - AT_i \quad (10)$$

B. Experimental Methodology

We use GridSim [18] to evaluate the performance of the proposed pricing function. 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].

We model our proposed utility-driven cluster RMS framework with the enhanced Economy-based Admission Control, Economy-based Resource Allocation and Job Control mechanisms that utilize the proposed user-level job submission specification and pricing function. This is referred to as Libra+$ in this section.

We also model two other resource allocation mechanisms: First-Come-First-Serve (FCFS) and Libra [11]. For FCFS, we model an existing cluster RMS that does not have admission control to decline jobs if their QoS requirements cannot be satisfied. The time-shared execution mechanism on its computation nodes assign equal shares of processing time among the executing jobs and thus do not enforce the required QoS of each job. We model FCFS to allocate a newly arrived job to the first computation node that finishes all its current executing jobs, based on the assumption that the estimated execution time is provided and is correct.

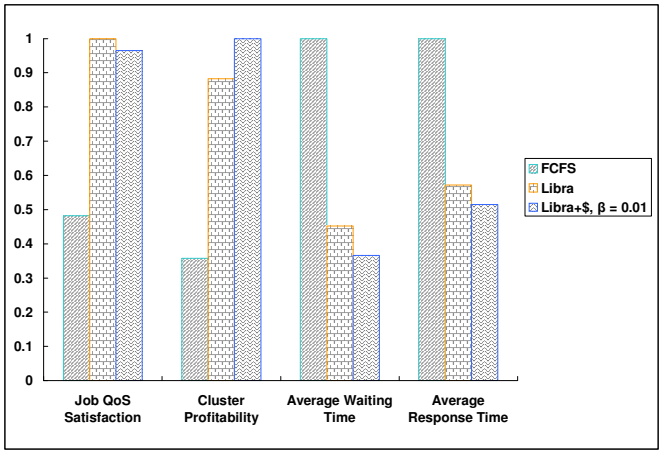


Fig. 2. Normalized comparison of FCFS, Libra, and Libra+$. Both Libra and Libra+$ achieve a substantially higher Job QoS Satisfaction and Cluster Profitability than FCFS. Similarly, both Libra and Libra+$ have a significantly lower Average Waiting and Response Time than FCFS. This shows the importance of considering and enforcing required QoS of each job.

In order to measure the Cluster Profitability metric, we also model FCFS 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 $C_i = EE_i * PBase_j$. FCFS is used for comparison so as to examine the benefits of considering and enforcing the required QoS of jobs using our proportional-share resource allocation based on required QoS (deadline) over traditional resource allocation mechanisms.

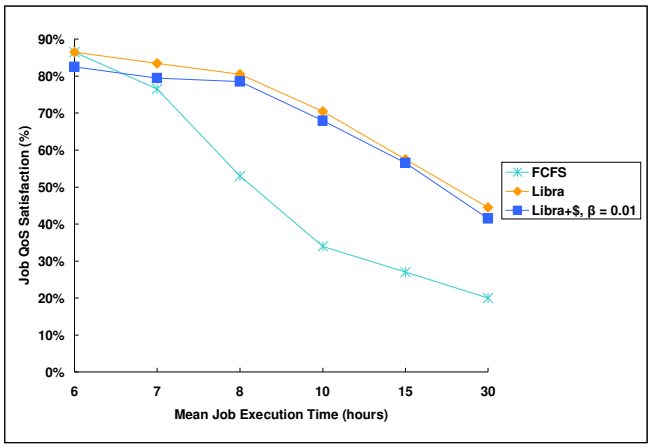
Libra [11] uses a pricing function that provides incentives for jobs with a more relaxed deadline to compute a static cost, so job i has its cost $C_i = \gamma * EE_i + \delta * EE_i / D_i$. γ is a factor for the first component that computes the cost based on the execution time of the job, so that longer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

For the cluster operating environment, we simulate a 13-node cluster called Manjra located at the University of Melbourne. The Manjra cluster has the following characteristics:

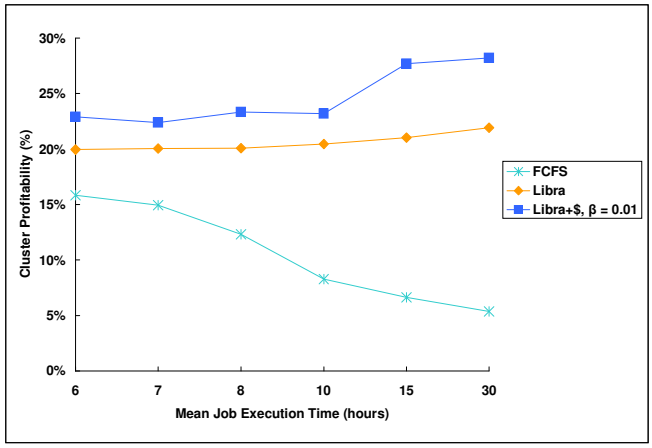
- SPEC rating: 684
- Number of computation nodes: 13
- Processor type on each computation node: INTEL Pentium4 2-GHz
- Operating System: Linux

For the experiments, we follow a similar experimental methodology in [21] to simulate the following cluster workload that models a high demand for cluster resources where the majority of jobs have short deadlines:

- 200 jobs with exponentially distributed job inter-arrival



(a) Job QoS Satisfaction



(b) Cluster Profitability

Fig. 3. Impact of increasing job execution time. An increasing mean job execution time results in both Libra and Libra+$ to have significantly higher Job QoS Satisfaction and Cluster Profitability over FCFS. Libra+$ generates increasing Cluster Profitability for decreasing Job QoS Satisfaction, demonstrating the effectiveness of its pricing function in improving the economic benefit of the cluster owner.

time of mean 0.5 hours and exponentially distributed job execution time E_i of mean 10 hours

- 80% of the 200 jobs belongs to a *high urgency* job class with a low $D_i / E_i = 1.5$ and a high $B_i / f(E_i) = 6$, where $f(E_i)$ is a function to compute the minimum budget required for job execution time E_i
- 20% of the 200 jobs belongs to a *low urgency* job class with a high $D_i / E_i = 6$ and a low $B_i / f(E_i) = 1.5$
- D_i and B_i are normally distributed within each high/low D_i/E_i and $B_i / f(E_i)$
- The high urgency and low urgency job classes are randomly distributed in arrival sequence
- For Libra+$, static pricing factor α = 1 and dynamic pricing factor $\beta = 0.01$

C. Evaluation of FCFS, Libra, and Libra+$

We evaluate the three resource allocation mechanisms: FCFS, Libra, and Libra+$ using the four performance evaluation metrics. Fig. 2 shows their normalized comparison.

For user-centric metrics, both Libra and Libra+$ are able to achieve substantially higher Job QoS Satisfaction and Cluster Profitability than FCFS since they consider the required QoS (deadline and budget) of each different job and allocate resources proportionally to each job based on the 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 as the cluster workload increases, thus denying jobs with insufficient budget.

On the contrary, the proposed pricing function still generates significantly higher profit than Libra even though fewer jobs are accepted, thus proving its effectiveness in improving the cluster owner's economic benefits. FCFS has the lowest Cluster Profitability as it does not consider and thus fail to fulfill the required QoS of most jobs.

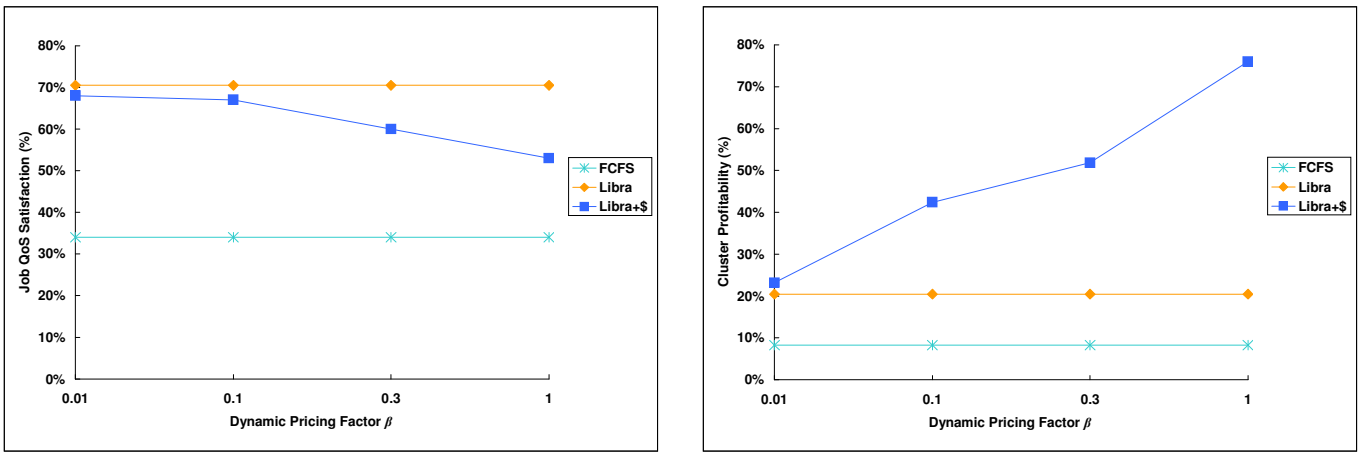
For system-centric metrics, both Libra and Libra+$ incurs significantly lower Average Waiting and Response Time than FCFS because their Economy-based Admission Control mechanisms consider the QoS constraints of jobs and filter jobs whose QoS constraints cannot be satisfied. Libra+$ has lower Average Waiting and Response Time than Libra since fewer jobs are accepted and executed.

D. Varying Cluster Workload

We examine the performance of Libra+$ with changing cluster workload. We increase the job execution time to represent increasing workload that would result in jobs failing to meet their required QoS (deadline).

Fig. 3(a) shows that FCFS has a significantly lower Job QoS Satisfaction than both Libra and Libra+$ with increasing mean job execution time. This demonstrates the importance of considering and enforcing required QoS of jobs and the effectiveness of implementing proportional-share resource allocation based on the required QoS (deadline) to satisfy more jobs. But, Libra+$ has a lower Job QoS Satisfaction than FCFS when the cluster workload is not high. For example, in Fig. 3(a), Libra+$ has a lower Job QoS Satisfaction than FCFS when mean job execution time is 6 hours. This is because Libra+$ declines some jobs due to insufficient budget.

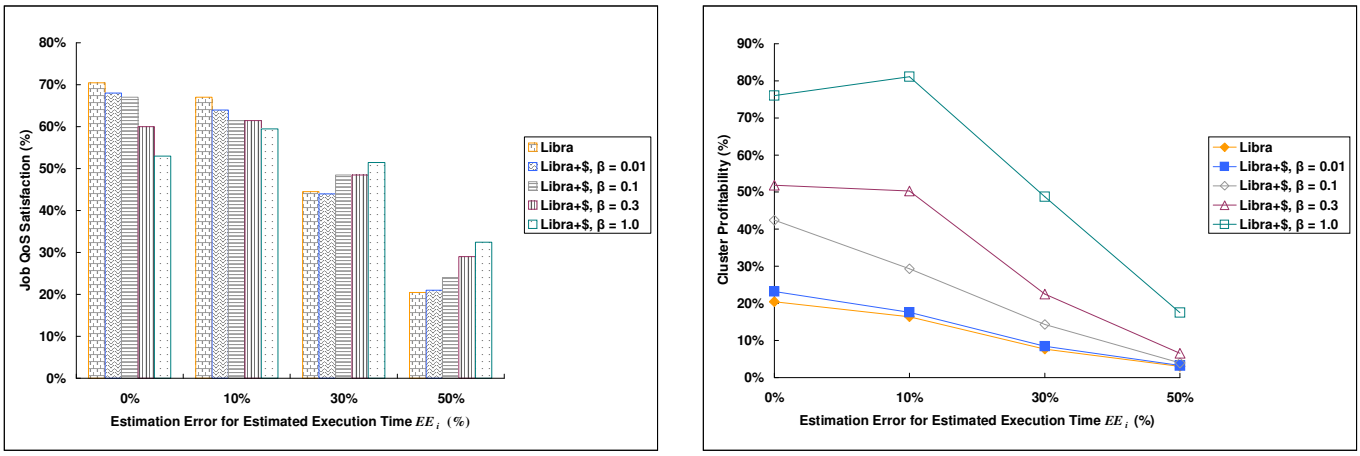
Fig. 3(b) shows that Libra+$ returns a considerably higher Cluster Profitability than Libra with increasing mean job execution time. This shows the effectiveness of the proposed pricing function in improving the economic benefit of the cluster owner even though Libra+$ accepts fewer jobs than Libra. As the cluster workload increases, only jobs that can afford the increased pricing are accepted by the Economy-based Admission Control mechanism. These fewer higher-priced jobs are able to maintain a higher Cluster Profitability to compensate for a lower Job QoS Satisfaction.



(a) Job QoS Satisfaction

(b) Cluster Profitability

Fig. 4. Impact of increasing dynamic pricing factor β. An increasing β for Libra+$ results in decreasing Job QoS Satisfaction, but increasing Cluster Profitability. The cluster owner can adjust the value of β to determine the level of sharing for the cluster.



(a) Job QoS Satisfaction

(b) Cluster Profitability

Fig. 5. Impact of increasing estimation error for estimated execution time EE_i. A higher dynamic pricing factor β for Libra+$ provides a higher level of tolerance against estimation errors for both Job QoS Satisfaction and Cluster Profitability.

E. 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.

Fig. 4(a) shows that an increasing β for Libra+$ results in decreasing Job QoS Satisfaction, while Fig. 4(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 are accepted. This is possible since jobs with sufficient budget are executed at a higher cost (due to higher β) to compensate for accepting fewer jobs due to insufficient budget. Another interesting point to note from Fig. 4(a) is that if β is set too high, the Job QoS Satisfaction can be lower than FCFS due to too many jobs having insufficient budget.

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). 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.

F. Tolerance against Estimation Error

We investigate the tolerance of Libra+$ against estimation error for estimated execution time EE_i. The estimation error is modeled as an under-estimated value of EE_i so as to examine the impact of delays caused by earlier jobs on later jobs. Delays in earlier jobs may result in later jobs to finish beyond their

deadlines, thus failing to meet their required QoS. For example, if we model an estimation error of 50%, then a job i whose execution time is 60 hours will therefore has an estimated execution time EE_i of 30 hours. We do not consider over-estimated value of EE_i since jobs accepted by the admission control will still be completed within their required deadline.

Fig. 5(a) shows that when there is no (0%) estimation error, a higher dynamic pricing factor β for Libra+$ results in a lower Job QoS Satisfaction. But, with increasing estimation error, a higher β results in a higher Job QoS satisfaction. This shows that a higher β provides a higher degree of tolerance against estimation errors since fewer jobs are accepted and thus the possibility of delays occurring on later jobs is lower. For example, in Fig. 5(a), β = 1.0 has the highest Job QoS Satisfaction for estimation error of more than 30%.

Fig. 5(b) shows that increasing estimation error results in lower Cluster Profitability as fewer jobs have their required QoS satisfied due to delays caused by earlier jobs. However, a higher β for Libra+$ can still achieve higher Cluster Profitability with increasing estimation error. This reiterates the effectiveness of the proposed pricing function in improving the economic benefit of the cluster owner.

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 under-estimated job execution times. Finally, the pricing function generates a higher economic benefit for the cluster owner.

Future work will investigate utility-driven resource allocation for more complex cluster application models, such as task-farming and parallel applications. Different pricing strategies based on economic models will also be examined.

Acknowledgment

We thank Srikumar Venugopal for his comments.

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 (HCW2001)*, 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 (CCGrid2002)*, 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 (PDPTA2000)*, 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] R. Buyya and M. Murshed, "GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1175–1220, Nov.–Dec. 2002.

[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 (AMS2000)*, 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 (PDPTA2002)*, Las Vegas, NV, June 2002.

[21] 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.