Service Level Agreement based Allocation of Cluster Resources: Handling Penalty to Enhance Utility

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

Jobs submitted into a cluster have varying requirements depending on user-specific needs and expectations. Therefore, in utility-driven cluster computing, cluster Resource Management Systems (RMSs) need to be aware of these requirements in order to allocate resources effectively. Service Level Agreements (SLAs) can be used to differentiate different value of jobs as they define service conditions that the cluster RMS agrees to provide for each different job. The SLA acts as a contract between a user and the cluster whereby the user is entitled to compensation whenever the cluster RMS fails to deliver the required service. In this paper, we present a proportional share allocation technique called LibraSLA that takes into account the utility of accepting new jobs into the cluster based on their SLA. We study how LibraSLA performs with respect to several SLA requirements that include: (i) deadline type whether the job can be delayed, (ii) deadline when the job needs to be finished, (iii) budget to be spent for finishing the job, and (iv) penalty rate for compensating the user for failure to meet the deadline.

1 Introduction

Clusters [15] have been rapidly utilized for an expanding range of applications that demand high-performance, high-throughput and high-availability computing services. They are not only used for computation-intensive applications, but also as replicated storage and backup facilities that provide essential fault tolerance and reliability.

The advent of *service-oriented Grid computing* [10] where geographically distributed resources such as clusters can be shared across various organizations, reinforces the importance for *utility-driven cluster computing*. Commercial vendors are now progressing aggressively towards providing a service market that provides dynamic service delivery where users only pay for what they use and thus save from investing heavily on computing facilities. Some examples include IBM's E-Business On Demand [2], HP's Adaptive Enterprise [1] and Sun Microsystem's pay-as-you-go [3].

Grid service brokers [22] and workflow engines [25] submit and monitor jobs with their service requirements via the local cluster RMS. To support utility-driven cluster computing, clusters need to differentiate utility or values for different service requests. *Service level agreements* (SLA) with precise *Quality of Service* (QoS) parameters need to be supported and enforced by the cluster RMS so as to fulfill the contractual obligations negotiated and agreed upon by both the cluster and users. In other words, the cluster RMS has to balance competing service requests, while ensuring that agreed levels of service performance are achieved.

However, existing cluster RMSs still adopt system-centric resource allocation approaches that maximize overall job performance and system usage, and thus are not ready to provide service-on-demand computing. On the other hand, market-based approaches [23] can support utility-driven computing where the *utility* or value is the monetary payment paid by the users for accessing the computing services. Using computational economy to support utility-driven resource allocation within clusters can regulate supply and demand of limited cluster resources at market equilibrium and differentiate service requests based on their utility.

Although market-based approaches have long been proposed, there are yet any actual implemented market-based RMSs that can demonstrate they work in practice due to the lack of enabling technologies. But, with numerous recent technological advances that can aid actual deployments of market-based RMSs [18], it is now timely to examine how market-based solutions can be applied effectively even though there still remains some key challenges [18] that need to be overcome first.

This paper focuses on a proportional share resource allocation technique called LibraSLA that applies market economy to achieve utility-driven cluster computing. The key contributions of this paper are:

- Defining a simple SLA with four basic QoS parameters: (i) deadline type whether the job can be delayed, (ii) deadline when the job needs to be finished, (iii) budget to be spent for finishing the job, and (iv) penalty rate for compensating the user for failure to meet the deadline. The penalty rate is represented by a linear penalty function that reduces the budget of the job over time after the lapse of its deadline.

- Developing an admission control and resource allocation mechanism that determines whether accepting a new job will enhance the aggregate utility of the cluster: The admission control examines how the new job will affect the SLA conditions of other accepted jobs, in particular how penalties incurred will decrease their utility. For resource allocation, LibraSLA allocates processing resources proportionally to job requests based on their deadline SLA property. In addition, LibraSLA allocates additional processing resources if available to the job with the highest return so as to achieve its utility faster.

- Analyzing the performance of LibraSLA based on varying SLA properties: (i) deadline type, (ii) deadline, (iii) budget, and (iv) penalty rate.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 outlines a simple SLA supporting four QoS parameters. Section 4 describes how LibraSLA examines and enforces SLA. Section 5 discusses performance evaluation results and Section 6 concludes this paper.

2 Related Work

Existing cluster RMSs such as Condor [21], LoadLeveler [11], Load Sharing Facility (LSF) [16], Portable Batch System (PBS) [5], and Sun Grid Engine (SGE) [19] still adopt system-centric approaches that optimize overall cluster performance. Cluster performance is often aimed at maximizing processor throughput and utilization for the cluster, and minimizing average waiting and response time for the jobs. They are thus not suitable for utility-driven cluster computing since they do not differentiate and thus neglect varying levels of utility or value that different cluster users have for each job request. Maui [20] is an advanced cluster scheduler that is designed to be highly configurable and extensible. It can be extended to build customized user-level schedulers that incorporate fine-grained policies and examine numerous resource allocation parameters such as QoS and advanced reservation. Currently, no market-based approaches have been designed for Maui to improve utility for either the cluster or users.

Numerous market-based approaches [23] have been proposed for resource management in parallel and distributed computing. REXEC [8] is a remote execution environment for a cluster of workstations that adopts market-based resource allocation. It assigns resources proportionally to jobs based on their users' bid (valuation) for each job. Tycoon [14] also adopts the same bid-based proportional share technique as REXEC, but extends it with continuous bids for allocating resources in a Grid of distributed clusters. In contrast, our LibraSLA prioritizes each job based on its SLA parameters that address two additional perspectives: (i) deadline when a job has to be finished and (ii) penalty rate to compensate the user if the deadline is not met. In addition, we aim to improve the aggregate utility of the cluster thru the consideration of penalties defined for respective SLA of different jobs.

Cluster-On-Demand [12] adopts distributed market-based task services to create a service market where penalties are incurred if jobs finish later than their required run times. It demonstrates the importance of balancing the reward against the risk of accepting and executing jobs, especially in the case of unbounded penalty. It also uses a discount rate based on present value to reduce future gains of a job in order to differentiate between delays in job execution. Similarly, our LibraSLA also consider penalties incurred on already accepted jobs by accepting a new job. But in our case, a job is penalized after the lapse of its deadline, instead of immediately after its run time. In addition, we also determine which job has higher return so that the job with the highest return is assigned additional resources if available to realize its utility faster. LibraSLA also studies resource allocation at a more fine-grained level as compared to Cluster-On-Demand. LibraSLA determines acceptance at the node level depending on available nodes within the cluster to execute a job. On the other hand, Cluster-On-Demand decides at the cluster level whether to accept a job into the cluster.

QoPS [13] is a QoS based scheduling technique for parallel jobs. It uses an admission control to guarantee the deadline of every accepted job by accepting a new job only if its deadline can be guaranteed without violating the deadlines of already accepted jobs. QoPS uses a slack factor for each job to represent the maximum delay that can be accommodated after its deadline. This allows earlier jobs with slack to be delayed if necessary so that future more urgent jobs can be accepted. On the other hand, our service model defines two types of deadlines: (i) hard deadline where the job has to be finished before the deadline and (ii) soft deadline where the job can finish anytime after the deadline. Instead of a slack factor, LibraSLA incorporates a SLA parameter called penalty rate to denote the user's flexibility with delays for soft deadlines through compensation. For the same job, a higher penalty rate means less flexibility than a lower penalty rate. Thus, LibraSLA attempts to minimize penalty to improve the cluster's aggregate utility. Another difference is that QoPS employs a kill-and-restart mechanism where a running job can be terminated to allow another job to be started so that a different schedule enables a new job to be accepted, while LibraSLA uses proportional share to vary the amount of resources for each job depending on their QoS needs.

Libra [17] is an earlier work done that successfully demonstrates that a market-based cluster scheduler is able to deliver more utility to users based on their QoS needs compared to traditional system-centric scheduling policies. Its market model is based on a commodity market [23] where Libra computes the price that users have to pay for their jobs be completed according to their QoS constraints. An enhanced pricing function [24] that is flexible, fair, dynamic and adaptive has also been proposed to improve the pricing scheme of the cluster so that the quoted price varies according to the workload of the cluster and prevents the cluster from overloading. LibraSLA incorporates a penalty function (thru the penalty rate parameter in the SLA) where the utility or value of the user will decrease over time after the deadline of the job has lapsed. Libra assumes that all jobs have hard deadlines and guarantees that accepted jobs will be finished within their hard deadlines. In contrast, LibraSLA allows jobs with soft deadlines to be delayed and compensated to accommodate jobs with hard deadlines. Finally, Libra only accepts jobs based on the workload of the cluster, whereas LibraSLA also examines the return of accepting each new job with respect to the current aggregate utility and workload of the cluster.

3 Service Level Agreement (SLA) for Utility-driven Cluster Computing

In utility-driven cluster computing, clusters provide computing services to users who perceive varying utility or value for completion of jobs. Clusters need to have knowledge of the types of service demanded by different users for each job in order to prioritize jobs according to user's needs.

Clusters should thus support SLA that provides a means for users and the cluster to agree upon the service quality to be offered. In other words, SLA acts as a contract that outlines obligations that both users and the cluster have to enforce and fulfill. For example, users have to pay for the service provided, while the cluster needs to be penalized to compensate users for failing to offer the required service. This also means that users can negotiate with the cluster about the service quality to be provided before accepting the SLA.

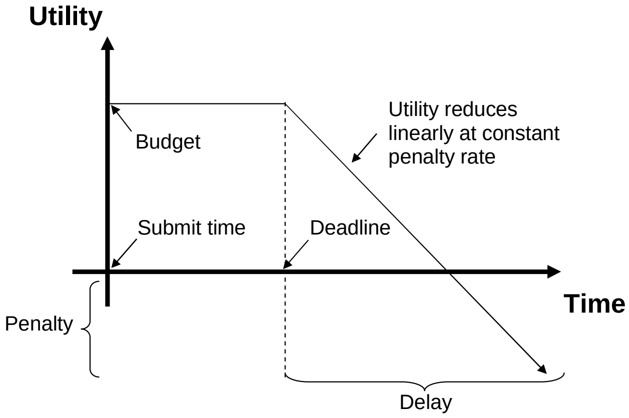


Figure 1. Impact of penalty function on utility.

We define a simple SLA for each job i that consists of four QoS parameters:

1. deadline type $deadline\_type_i$: A deadline can be *hard* or *soft*. A hard deadline denotes that the user wants the job to be finished before the deadline, whereas a soft deadline means that the user can accommodate delay. However, for soft deadline, the delay can be unbounded depending on the penalty rate of the job. Therefore, the user can give an appropriate penalty rate value to possibly limit the maximium delay.

2. deadline $deadline_i$: The time period in which the job needs to be finished.

3. budget $budget_i$: The maximum amount of currency that the user is willing to pay for the job to be completed.

4. penalty rate $penalty\_rate_i$: The penalty rate penalizes the cluster to compensate the user for failure to meet the deadline of the job. It also reflects the user's flexibility with delayed deadline of the job. A higher penalty rate limits the delay to be shorter than that of a lower penalty rate.

The key aspect of our SLA is that it incorporates a *penalty* function (thru the penalty rate QoS parameter). This is realistic as users need to have assurance that their required services will be maintained by the cluster. The penalty function not only penalizes the cluster for its service failure, but compensates the users for tolerating the service failure. For simplicity, we model the penalty function as linear, as in other previous works [9][12]. Our penalty function reduces the budget of the job over time after the lapse of its deadline, rather than after its run time in [12]. Figure 1 shows the impact of the penalty function on utility.

For each job i, the cluster achieves a utility $utility_i$ depending on its penalty rate $penalty\_rate_i$ and delay $delay_i$:

$$utility_i = budget_i - (delay_i * penalty\_rate_i) \qquad (1)$$

Job i has delay $delay_i$ if it takes longer to complete than its given deadline $deadline_i$:

$$delay_i = (finish\_time_i - submit\_time_i) - deadline_i \qquad (2)$$

where $submit\_time_i$ is the time when job i is submitted into the cluster and $finish\_time_i$ is the time when job i is completed. So, Job i has no delay (ie. $delay_i \leq 0$) if it completes before the deadline, with the cluster achieving the full budget $budget_i$ as utility $utility_i$. If there is a delay (ie. $delay_i > 0$), $utility_i$ drops linearly till it becomes negative and transform into a penalty (ie. $utility_i < 0$) after it exceeds $budget_i$.

Since penalties are unbounded depending on the penalty rate and delay of each job, the cluster needs to be careful about accepting new jobs into the cluster. Failure to deliver the required service due to accepting too many jobs may result in heavily penalized jobs that can dramatically erode previously achieved utility. Therefore, we develop a SLA based proportional share technique called LibraSLA (described in section 4) that considers the risk of penalties to improve aggregate utility for the cluster.

In addition to the SLA requirements, LibraSLA considers the following job details:

1. run time $runtime_i$: The run time for job i is the time period required to complete job i on a computation node provided that it is allocated the node's full proportion of processing power. Thus, the run time varies on nodes of different hardware and software architecture and does not include any waiting time and communication latency. The run time may also be expressed in terms of the job length in million instructions (MI).

2. number of processors $numproc_i$: The number of processors requested by job i. A sequential job will need only a single processor (ie. $numproc_i = 1$), while a parallel job will request multiple processors (ie. $numproc_i > 1$).

4 SLA Based Proportional Share with Utility Consideration

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

- Users express utility as the budget or amount of real money (as in the human world) they are willing to pay for the service. Real money is a well-defined currency [18] that will promote resource owner and user participation in distributed system environments. A user's budget is limited by the amount of currency that he has which may be distributed and administered through monetary authorities such as GridBank [6]. Since our focus is on resource allocation (which job to accept and which nodes to execute the accepted job), we do not venture further into other market concepts such as user bidding strategies and auction pricing mechanisms.

- Users only gain utility and thus pay for the service (based on QoS parameters in SLA) upon the completion of their jobs. For simplicity, the user pays the full budget for a job to the cluster if its deadline is fulfilled. But, if a job is delayed, the cluster will achieve a reduced utility or incur a penalty depending on length of the delay.

- The estimated run time of each job provided during job submission is accurate. Estimated run times can be predicted in advance based on means such as past execution history.

- The deadline given by a user for the job must be more than its run time; otherwise it is not accepted into the cluster.

- The SLA of a job does not change after job acceptance. This means that the QoS parameters specified by the user during job submission do not change after the job is accepted.

- Users can only submit jobs into the cluster through the cluster RMS. This means that the cluster RMS has full knowledge of allocated workload currently in execution and remaining available resources on each computation node.

- The computation nodes can be homogeneous (have the same hardware architectures) or heterogeneous (have different hardware architectures). In the case of heterogeneous computation nodes, the estimated run time needs to be converted to its equivalent on the allocated computation node.

- The operating system at each computation node uses time-shared scheduling support where multiple processor time partitions can be assigned to different jobs.

Bid-based proportional share [8][14] allocates proportions of a resource such as processor time to users based on their bids (budget QoS parameter in SLA) for each job. In other words, the resource share assigned to a job is proportional to the user's bid value in comparison to other users' bids. However, this approach does not take into consideration the characteristics of the job and its other essential SLA properties such as deadline and penalty rate.

To address this for SLA support, LibraSLA adopts the proportional share approach in Libra [17] that allocates processor time share $share_{ij}$ to job i on node j based on its remaining run time $remain\_runtime_{ij}$ and remaining deadline QoS $remain\_deadline_{ij}$, rather than users' bids $budget_i$:

$$share_{ij} = \frac{remain\_runtime_{ij}}{remain\_deadline_{ij}} \qquad (3)$$

where initially, $remain\_runtime_{ij} = runtime_i$ and $remain\_deadline_{ij} = deadline_i$. $share_{ij}$ also denotes the minimum share that is required by job i in order to enforce its deadline $deadline_i$. Proportional share based on deadline not only allows more jobs to be accepted (since the allocated processor time shares are spread across the deadlines), but also ensures that their deadlines are met.

Therefore, the total processor time share $total\_share_j$ required to meet all deadlines of n_j jobs allocated to node j is:

$$total\_share_j = \sum_{i=1}^{n_j} share_{ij} \qquad (4)$$

Delays occur when $total\_share_j$ exceeds the maximum processor time that node j can offer.

4.1 Computing Return for Jobs and Nodes

In order to improve aggregate utility for the cluster, LibraSLA needs to consider the utility of each job to determine which job has a higher return. The return $return_{ij}$ of a job i allocated to run on node j is computed as:

$$return_{ij} = utility_{ij}/runtime_i/deadline_i \qquad (5)$$

Recall that it is possible for a job i allocated to node j to have negative utility (ie. $utility_{ij} < 0$), also known as a penalty as defined in (1). Thus, in this case, job i will also have negative return (ie. $return_{ij} < 0$).

LibraSLA regards jobs that have shorter deadlines for an expected utility per unit of run time $(utility_{ij}/runtime_i)$ to have a higher return. Jobs with shorter deadlines require a shorter commitment period as compared to those with longer deadlines. Thus, it increases the flexibility of accepting later arriving but possibly jobs with higher return as a full schedule of jobs with long deadlines may result in these future jobs being blocked by the admission control.

Jobs with shorter deadlines are also penalized more heavily than those with longer deadlines. This discourages accepting more jobs that can delay other accepted urgent jobs and jeopardize the cluster's aggregate utility.

LibraSLA can thus compute the return $return_j$ of node j to determine whether node j is improving the aggregate utility of the cluster or not:

$$return_j = \sum_{i=1}^{n_j} return_{ij} \qquad (6)$$

$return_j$ also gives an indication of whether node j is overloaded with too many jobs and failing to satisfy their SLA. $return_j$ will be lower when the workload on node j is higher since insufficient resources will result in jobs being delayed and thus having lower utility.

4.2 Admission Control and Resource Allocation

Algorithm 1: Pseudo-code for admission control and resource allocation of LibraSLA.

```
for $j \leftarrow 0$ to $m - 1$ do
    add job $new$ temporarily into $ListJobs_j$ ;
    $new\_return_j \leftarrow$ compute_new_return ($ListJobs_j$);
    remove job $new$ from $ListJobs_j$ ;
    if $new\_return_j \geq return_j$ then
        if $deadline\_type_{new}$ is SOFT then
            place node $j$ in $ListHigherReturnNodes_{new}$ ;
        else if $deadline\_type_{new}$ is HARD and $delay_{new} \leq 0$ then
            place node $j$ in $ListHigherReturnNodes_{new}$ ;
        endif
    endif
endfor
if $ListHigherReturnNodes_{new}\_size \geq numproc_{new}$ then
    sort $ListHigherReturnNodes_{new}$ by $new\_return_j$ in descending order;
    for $j \leftarrow 0$ to $numproc_{new} - 1$ do
        allocate job $i$ to node $j$ of $ListHigherReturnNodes_{new}$ ;
    endfor
else
    reject job $new$;
endif
```

Since the SLA of each job incorporates a penalty function (as described in section 3), LibraSLA implements an admission control to ensure that more utility is achieved, instead of less utility due to accepting too many jobs and failing to meet their deadlines. The admission control determines whether a new job new should be accepted into the cluster depending on:

- $numproc_{new}$: A new job is not accepted if there are not enough available processors to run it. This often happens to parallel jobs as they require more processors.

- $deadline\_type_{new}$: If the new job requires hard deadline and there are no nodes that can fulfill its deadline, then it is not accepted.
- $return_j$: This denotes whether each node j will increase or decrease the aggregate utility if it is allocated this new job. Therefore, a new job can be accepted into the cluster depending on the return of each individual node.

Algorithm 1 shows how LibraSLA decides whether to accept a new job based on nodes with the highest return. Assuming that the cluster has m nodes, LibraSLA first determines the return of each node (using compute_new_return function in Algorithm 2) for accepting the new job new (line 2–4). A node is suitable if it has higher return after accepting the new job and can satisfy its hard deadline if required (line 5–11). The new job is then accepted if there are enough suitable nodes as requested (line 13) and allocated to the node with the highest return (line 14–17).

Algorithm 2: Pseudo-code for compute_new_return($ListJobs_j$) function.

```
1  $new\_return_j \leftarrow 0$;
2  $total\_share_j \leftarrow 0$;
3  set first job in $ListJobs_j$ to be job with the highest return;
4  for $i \leftarrow 0$ to $ListJobs_j\_size - 1$ do
5      $total\_share_j \leftarrow total\_share_j + share_{ij}$;
6      $return_{ij} \leftarrow budget_i / runtime_i / deadline_i$;
7      if job $i$ has higher return then
8          set job $i$ to be job with the highest return;
9      endif
10 endfor
11 if $total\_share_j \geq$ maximum processor time of node $j$ then
12     increase share of job with the highest return by remaining
       unallocated processor time;
13     for $i \leftarrow 0$ to $ListJobs_j\_size - 1$ do
14         $new\_return_j \leftarrow$
           $new\_return_j + budget_i / runtime_i / deadline_i$;
15     endfor
16 else
17     for $i \leftarrow 0$ to $ListJobs_j\_size - 1$ do
18         if job $i$ is job with the highest return or job $i$ has
           HARD deadline then
19             $new\_return_j \leftarrow new\_return_j +$
               $budget_i / runtime_i / deadline_i$;
20         else
21             decrease share of job $i$ by shortfall proportion of
               processor time;
22             compute $delay_i$;
23             compute $utility_{ij}$;
24             $new\_return_j \leftarrow new\_return_j +$
               $utility_i / runtime_i / deadline_i$;
25         endif
26     endfor
27 endif
28 return $new\_return_j$;
```

Algorithm 2 computes the new return of a node for accepting a new job. It first determines total processor time share required to fulfill the deadlines of its allocated jobs plus the new job (line 5). It also identifies the job with the highest return based on the budget (line 6–9). If there is any remaining unallocated processor time, the job with the highest return is given this additional remaining share to realize its utility faster (line 11–12). In this case, the return of the node is computed with the utility of each job same as its budget (line 13–15). If there is insufficient processor time, only the job with the highest return and jobs with hard deadlines are not delayed (line 18–19), while the other jobs with soft deadlines shares the shortfall processor time proportionally (line 21). The return of these delayed jobs is then computed accordingly (line 22–24).

5 Performance Evaluation

In this section, we discuss and evaluate the performance of LibraSLA. We first explain our experimental methodology, followed by detailed performance analysis of LibraSLA with respect to varying SLA properties: (i) deadline type, (ii) deadline, (iii) budget, and (iv) penalty rate.

5.1 Experimental Methodology

We use GridSim [7] to simulate a cluster RMS environment that utilizes LibraSLA for resource allocation. Our experiments employ real workload trace from Feitelson's Parallel Workload Archive [4]. The selected subset of the last 1000 jobs in the SDSC SP2 trace from April 1998 to April 2000 has the following properties:

- Average inter arrival time: 2276 seconds (37.93 minutes)
- Average run time: 10610 seconds (2.94 hours)
- Average number of allocated processors: 18

The 128-node IBM SP2 located at San Diego Supercomputer Center (SDSC) has the following characteristics:

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

The real workload trace does not contain any information about users' SLA parameters including deadline type, deadline, budget, and penalty rate. But, results are dependent on distributions of these four QoS parameters as they determine how LibraSLA allocate resources to jobs. Therefore, we follow a similar experimental methodology in Cluster-On-Demand [12] to represent SLA properties for the workload:

- 20% of the jobs belongs to a *high urgency* job class with a hard deadline of low $deadline_i/runtime_i$, a high $budget_i/f(runtime_i)$ and a high $penalty\_rate_i/g(runtime_i)$, where $f(runtime_i)$ and $g(runtime_i)$ are functions to represent the minimum budget and penalty rate that the user will quote with respect to $runtime_i$.

- 80% of the jobs belongs to a *low urgency* job class with a soft deadline of high $deadline_i/runtime_i$, a low $budget_i/f(runtime_i)$ and a low $penalty\_rate_i/g(runtime_i)$.

- The *deadline high:low ratio* refers to the ratio of the means for high $deadline_i/runtime_i$ and low $deadline_i/runtime_i$, likewise for *budget high:low ratio* and *penalty rate high:low ratio*. For the experiments, the jobs have a *deadline high:low ratio* of 7, a *budget high:low ratio* of 7, and a *penalty rate high:low ratio* of 4.

- Values are normally distributed within each high and low $deadline_i/runtime_i$, $budget_i/f(runtime_i)$ and $penalty\_rate_i/g(runtime_i)$ respectively.

- The high urgency and low urgency job classes are randomly distributed in arrival sequence.

Our performance evaluation examines the relative performance of LibraSLA with respect to Libra under varying cluster workload for the following SLA properties: (i) deadline type (section 5.3), (ii) deadline (section 5.4), (iii) budget (section 5.5), and (iv) penalty rate (section 5.6). Libra does not differentiate between hard and soft deadlines, thus accepting a new job only if there are sufficient nodes as requested to fulfill its deadline. Another key difference is that Libra selects nodes based on the best fit strategy. In other words, nodes that have the least available processor time after accepted the new job will be selected first. This ensures that nodes are saturated to their maximum so that more later arriving jobs may be accepted.

To demonstrate the effectiveness of resource allocation, we need to model a heavy workload scenario where the demand for cluster resources exceeds the supply. We use *arrival delay factor* with the inter arrival time of jobs (available from the trace) to model the cluster workload. For example, an arrival delay factor of 0.01 means that a job with 400 seconds of inter arrival time from the trace now has a simulated inter arrival time of 4 seconds. Thus, an increasing arrival delay factor represents decreasing workload.

We use a mean factor to denote the mean value for the normal distribution of deadline, budget and penalty rate SLA parameters. A mean factor of 2 represents having mean value double than that of 1 (ie. higher).

5.2 Overview of Performance Results

Generally, the improvement of LibraSLA over Libra decreases as the arrival delay factor increases. This is because Libra can also complete more jobs and achieve more utility when the workload is not heavy (higher arrival delay factor). Thus, we are able to demonstrate that LibraSLA is effective in differentiating jobs with higher utility in heavy workload situations.

5.3 Impact of Deadline Type

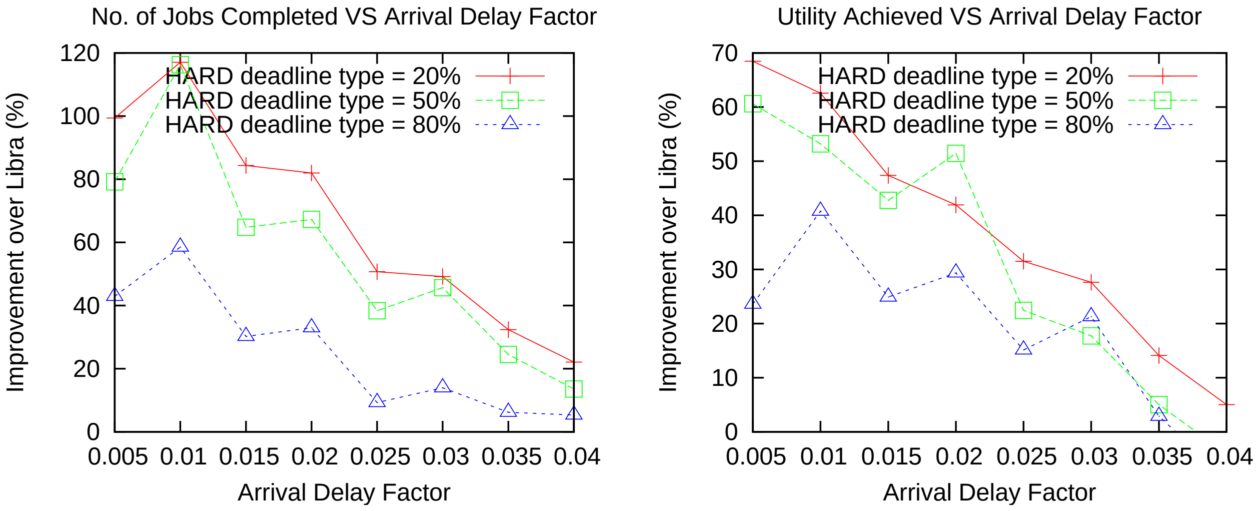
We vary the proportion of jobs belonging to the high urgency job class with hard deadlines at 20%, 50%, and 80% to examine the impact of deadline type on LibraSLA. Figure 2 shows that when there are more jobs with hard deadline (eg. 80%), the improvement over Libra is lower since there are less jobs with soft deadline to accommodate the required delays without risking the aggregate utility achieved. We can see that on average LibraSLA completes about 20% more jobs (Figure 2(a)) and achieves 10% more utility (Figure 2(b)) than Libra when there are 20% jobs with hard deadline as compared to 80%.

5.4 Impact of Deadline

We study how LibraSLA performs for different deadlines using the deadline mean factor of 1, 2, and 3. Figure 3 shows that LibraSLA has a substantial large improvement over Libra for a deadline mean factor of 1 when the arrival delay factor is lower (0.005–0.025). This is because LibraSLA determines utility based on the deadline of the jobs, and thus can differentiate jobs with high utility and short deadline when the workload is high. However, with higher deadline mean factor of 2 and 3, the improvement over Libra is lower over increasing arrival delay factor since Libra is also able to complete more jobs with longer deadlines. We can also see that LibraSLA has a more constant improvement over Libra for sufficiently long deadlines (deadline mean factor of 2 and 3) as opposed to short deadlines (deadline mean factor of 1). This highlights that the deadline QoS has a strong impact on the performance of LibraSLA.

5.5 Impact of Budget

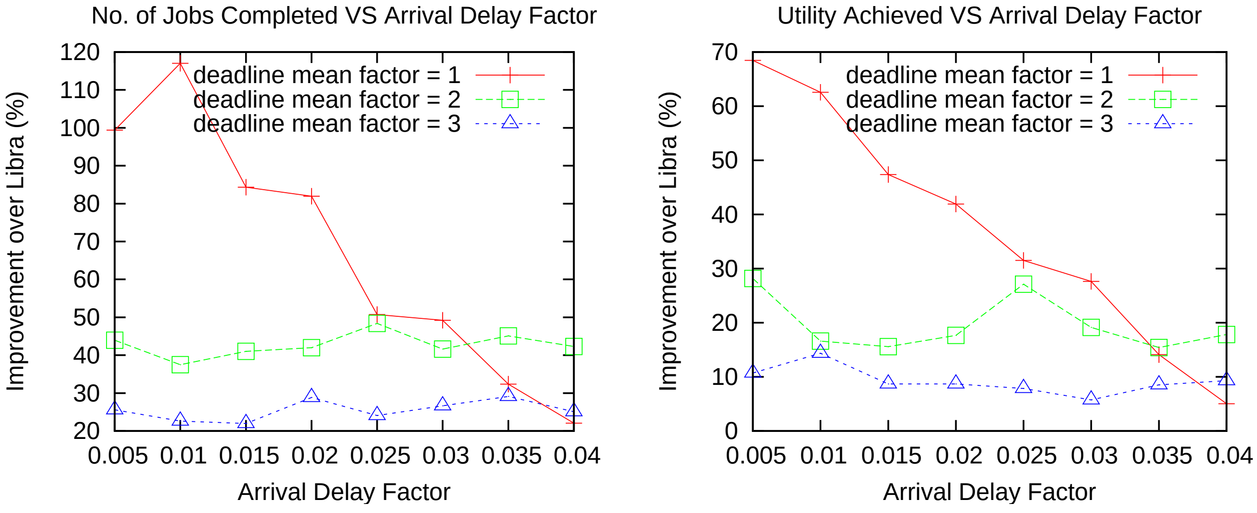
We investigate the performance of LibraSLA for different budgets with budget mean factor of 1, 2, and 3. Figure 4(a) shows that LibraSLA completes more jobs than Libra for higher budget mean factors. When jobs have higher budgets, LibraSLA can accommodate more soft deadline jobs with delays which in turn improves the aggregate utility (Figure 4(b)). However, the utility improvement also



(a) No. of Jobs Completed

(b) Utility Achieved

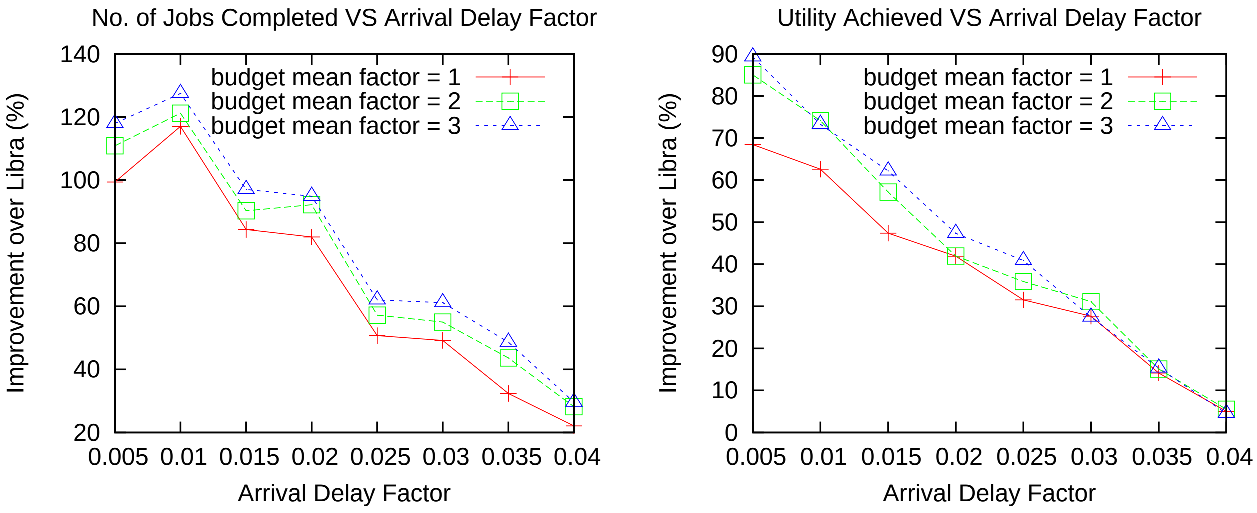
Figure 2. Impact of deadline type on increasing workload.



(a) No. of Jobs Completed

(b) Utility Achieved

Figure 3. Impact of deadline mean factor on increasing workload.



(a) No. of Jobs Completed

(b) Utility Achieved

Figure 4. Impact of budget mean factor on increasing workload.

diminishes with increasing arrival delay factor (0.03–0.04) as Libra can also accept more jobs to increase utility.

5.6 Impact of Penalty Rate

We observe how LibraSLA performs with changing penalty rate mean factor of 1, 2, and 3. Figure 5 shows that LibraSLA has less improvement over Libra for increasing penalty rate mean factor. With higher penalty rate, LibraSLA is limited to accepting fewer jobs in order to preserve the aggregate utility. This explains the lower improvement in utility achieved as well since jobs with soft deadlines now has higher penalty rate and can potentially risk the aggregate utility.

6 Conclusion

This paper has presented an approach to handle penalties incorporated in SLAs in order to enhance the utility of the cluster. We have also outlined a basic SLA with four QoS parameters: (i) deadline type, (ii) deadline, (iii) budget, and (iv) penalty rate, before describing a proportional share technique called LibraSLA that considers these QoS parameters. Simulation results show that LibraSLA performs better than Libra by accommodating more jobs thru soft deadlines and minimizing penalties. This work has thus reinforced the need to employ and consider SLAs in cluster-level resource allocation in order to support utility-driven cluster computing for service-oriented Grid computing.

References

[1] HP Grid Computing,
http://www.hp.com/techservers/grid, May 2005.

[2] IBM Grid Computing,
http://www.ibm.com/grid, May 2005.

[3] Sun Microsystems Utility Computing,
http://www.sun.com/service/utility, May 2005.

[4] Parallel Workloads Archive,
http://www.cs.huji.ac.il/labs/parallel/workload, May 2005.

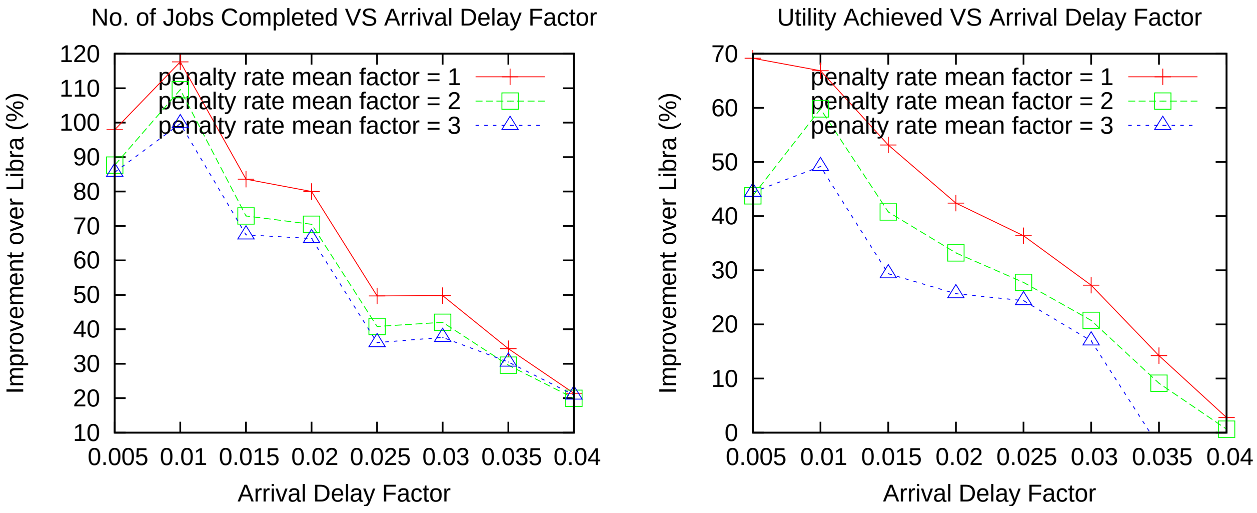
[5] Altair Grid Technologies. *OpenPBS Release 2.3 Administrator Guide*, Aug. 2000.

[6] A. Barmouta and R. Buyya. GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration. In *Proceedings of the 3rd Workshop on Internet Computing and E-Commerce (ICEC 2003), International Parallel and Distributed Processing Symposium (IPDPS 2003)*, Nice, France, Apr. 2003.

[7] 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*, 14(13–15):1175–1220, Nov.–Dec. 2002.

[8] B. N. Chun and D. E. Culler. Market-based Proportional Resource Sharing for Clusters. Technical Report CSD-1092, Computer Science Division, University of California at Berkeley, Jan. 2000.

[9] B. N. Chun and D. E. Culler. User-centric Performance Analysis of Market-based Cluster Batch Schedulers. In *Proceedings of the 2nd International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, pages 22–30, Berlin, Germany, May 2002.



(a) No. of Jobs Completed

(b) Utility Achieved

Figure 5. Impact of penalty rate mean factor on increasing workload.

[10] I. Foster and C. Kesselman, editors. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, CA, 2003.

[11] IBM. *LoadLeveler for AIX 5L Version 3.2 Using and Administering*, Oct. 2003.

[12] D. E. Irwin, L. E. Grit, and J. S. Chase. Balancing Risk and Reward in a Market-based Task Service. In *Proceedings of the 13th International Symposium on High Performance Distributed Computing (HPDC13)*, pages 160–169, Honolulu, HI, June 2004.

[13] M. Islam, P. Balaji, P. Sadayappan, and D. K. Panda. Towards Provision of Quality of Service Guarantees in Job Scheduling. In *Proceedings of the 6th International Conference on Cluster Computing (CLUSTER 2004)*, pages 245–254, San Diego, CA, Sept. 2004.

[14] K. Lai, L. Rasmusson, E. Adar, S. Sorkin, L. Zhang, and B. A. Huberman. Tycoon: an Implementation of a Distributed Market-based Resource Allocation System. Technical Report cs.DC/0412038, HP Labs, Palo Alto, Dec. 2004.

[15] G. F. Pfister. *In Search of Clusters*. Prentice Hall PTR, Upper Saddle River, NJ, second edition, 1998.

[16] Platform Computing. *LSF Version 4.1 Administrator's Guide*, 2001.

[17] 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*, 34(6):573–590, May 2004.

[18] J. Shneidman, C. Ng, D. C. Parkes, A. AuYoung, A. C. Snoeren, A. Vahdat, and B. N. Chun. Why Markets Could (But Don't Currently) Solve Resource Allocation Problems in Systems. In *Proceedings of the 10th Workshop on Hot Topics in Operating Systems (HotOS X)*, Santa FE, NM, June 2005.

[19] Sun Microsystems. *Sun ONE Grid Engine, Administration and User's Guide*, Oct. 2002.

[20] Supercluster Research and Development Group. *Maui Scheduler Version 3.2 Administrator's Guide*, 2004.

[21] University of Wisconsin-Madison. *Condor Version 6.7.1 Manual*, 2004.

[22] S. Venugopal, R. Buyya, and L. Winton. A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids. In *Proceedings of the 2nd International Workshop on Middleware for Grid Computing (MGC 2004)*, pages 75–80, Toronto, Canada, Oct. 2004.

[23] C. S. Yeo and R. Buyya. A taxonomy of market-based resource management systems for utility-driven cluster computing. Technical Report GRIDS-TR-2004-12, Grid Computing and Distributed Systems (GRIDS) Laboratory, University of Melbourne, Melbourne, Australia, Dec. 2004.

[24] C. S. Yeo and R. Buyya. Pricing for Utility-driven Resource Management and Allocation in Clusters. In *Proceedings of the 12th International Conference on Advanced Computing and Communications (ADCOM 2004)*, pages 32–41, Ahmedabad, India, Dec. 2004.

[25] J. Yu and R. Buyya. A Novel Architecture for Realizing Grid Workflow Using Tuple Spaces. In *Proceedings of the 5th International Workshop on Grid Computing (GRID 2004)*, pages 119–128, Pittsburgh, PA, Nov. 2004.