Integrated Risk Analysis for a Commercial Computing Service in Utility Computing

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

March 12, 2008

Abstract. Recent technological advances in grid computing enable the virtualization and dynamic delivery of computing services on demand to realize utility computing. In utility computing, computing services will always be available to the users whenever the need arises, similar to the availability of real-world utilities, such as electrical power, gas, and water. With this new outsourcing service model, users are able to define their service needs through Service Level Agreements (SLAs) and only have to pay when they use the services. They do not have to invest on or maintain computing infrastructures themselves and are not constrained to specific computing service providers. Thus, a commercial computing service will face two new challenges: (i) what are the objectives or goals it needs to achieve in order to support the utility computing model, and (ii) how to evaluate whether these objectives are achieved or not. To address these two new challenges, this paper first identifies four essential objectives that are required to support the utility computing model: (i) manage wait time for SLA acceptance, (ii) meet SLA requests, (iii) ensure reliability of accepted SLA, and (iv) attain profitability. It then describes two evaluation methods that are simple and intuitive: (i) separate and (ii) integrated risk analysis to analyze the effectiveness of resource management policies in achieving the objectives. Evaluation results based on simulation successfully demonstrate the applicability of separate and integrated risk analysis to assess policies in terms of the objectives. These evaluation results which constitute an a posteriori risk analysis of policies can later be used to generate an a priori risk analysis of policies by identifying possible risks for future utility computing situations.

Keywords: utility computing, grid economy, risk, service, resource management, performance evaluation

1. Introduction

With the advance of parallel and distributed technologies, such as cluster computing [20][3] and grid computing [9] that enable on-demand resource sharing across various organizations, commercial vendors such as Amazon [2], HP [10], IBM [11], and Sun Microsystems [27] are now progressing aggressively towards realizing the next era of computing model – *utility computing* [36]. The vision of utility computing is to provide computing services whenever users want them. Users no longer have to invest heavily on or maintain their own computing infrastructure. Instead, they just have to pay for what they use whenever they want by outsourcing jobs to dedicated commercial computing services for completion. This means that users define their service needs and expect them to be delivered by commercial computing service providers. Thus, a commercial computing service will face two new challenges: (i) what are the objectives or goals it needs to achieve in order to support the utility computing model, and (ii) how to evaluate whether these objectives are achieved or not.

Different users have distinctive needs for various jobs and thus demand specific Quality of Service (QoS) for completing these jobs. A user can negotiate the QoS terms and conditions with a commercial computing service provider before formally outlining the confirmed negotiations in a Service Level Agreement (SLA). The SLA acts as an official contract for the computing service to deliver the expected QoS to the user.

To support the utility computing model, a commercial computing service has to solicit resource and QoS requirements from the user to decide whether to accept a service request or not. Resource requirements specify what is essential to run and complete the job successfully, such as the number of processors, memory size, disk storage size, and runtime estimate (estimated time taken to complete the job). QoS requirements state what is necessary to realize the user's service target. QoS attributes that can be specified in a SLA include time, cost, reliability, and/or trust/security [32]. In this paper, we consider two of these QoS attributes: (i) deadline that a job should be completed in (as time QoS attribute) and (ii) budget that the user is willing to pay the commercial computing service for completing the job (as cost QoS attribute).

This paper focuses on evaluating suitable resource management policies for a commercial computing service to support utility computing based on its objectives. As there are numerous policies [17][19][13][22][31][32] available, it is non-trivial to identify the best policy that truly meets the objectives. Therefore, the key contributions of this paper are to:

- Identify four essential objectives for a commercial computing service to support the utility computing model: (i) manage wait time for SLA acceptance, (ii) meet SLA requests, (iii) ensure reliability of accepted SLA, and (iv) attain profitability.

- Develop two evaluation methods that are simple and intuitive: (i) separate and (ii) integrated risk analysis to analyze the effectiveness of resource management policies in achieving the objectives.

- Provide comprehensive performance analysis of various policies through trace-based simulation to reveal the best policy in achieving different objectives for two economic models: (i) commodity market model and (ii) bid-based model.

Evaluation results based on simulation demonstrate that both separate and integrated risk analysis can be applied successfully to evaluate resource management policies in terms of achieving the objectives. These evaluation results which constitute an a posteriori risk analysis of policies can later be used to generate an a priori risk analysis of policies by identifying possible risks for future utility computing situations. Thus, commercial computing service providers are now able to identify and implement ideal policies to realize utility computing.

This paper is organized as follows: Section 2 discusses related work. Section 3 identifies four essential objectives that a commercial computing service needs to achieve in utility computing and how they can be measured. Section 4 describes how separate and integrated risk analysis can determine whether various resource management policies can achieve the objectives. Section 5 describes the evaluation methodology and simulation setup to assess these policies. Section 6 compares the performance of various policies with regards to the objectives. Section 7 concludes.

2. Related Work

Numerous resource management systems [29][14][21][1][26] are available to provide different policies to allocate jobs. However, new service parameters need to be considered and enforced for utility computing, such as the deadline to complete the job, the budget the user will pay for its completion, and the penalty for any deadline violation. Therefore, several new works [24][12][13][22][31][33][35] proposes policies using admission control to support quality-driven computing services by selectively accepting new jobs based on certain service parameters. Admission control helps to maintain the level of service when there is only a limited supply of computing resources to meet an unlimited demand of service requests. Hence, when the demand is higher than the supply of resources, a computing service needs to either reject new service requests to ensure previously accepted requests are not affected or compromise previously accepted service requests to accommodate new requests. But, there is no proposed work to evaluate the effectiveness of these service-oriented policies in the context of utility computing, in particular the ability to satisfy essential objectives of a commercial computing service.

Various other works [15][16][12][22] have addressed some form of risk in computing jobs. In [12] and [22], the risk of paying penalties to compensate users is minimized so as not to reduce the profit of service providers. Computation-at-Risk (CaR) [15][16] determines the risk of completing jobs later than expected based on either the makespan (response time) or the expansion factor (slowdown). GridIS [31] shows that a conservative provider earns much less profit due to accepting too few jobs to run, as compared to an aggressive provider who earn more profit even though more jobs result in deadline violations. However, none of these works consider and model the impact of policies on the achievement of objectives as risks.

Our work is inspired by service management [30] and risk management [6] in the field of economics which has been widely studied, adopted and proven. From the economics perspective, a commercial computing service in utility computing is a business intended to sell services to consumers and generate profit from them. Comprehensive studies in service management [23] have shown that customer satisfaction is a crucial success factor to excel in the service industry. Customer satisfaction affects customer loyalty, which in turn may lead to referrals of new customers [30]. These achievements thus constitute the sustainability and improvement of revenue for a business. Therefore, we apply similar service quality factors for the proposed three user-centric objectives to ensure that customer satisfaction is achieved. In addition, economists have proposed enterprise risk management [18] to manage the risks of a business based on its targeted objectives. Hence, we adopt a similar approach to evaluate resource management policies of a commercial computing service with respect to its objectives using separate and integrated risk analysis.

This paper is the revised version of a preliminary paper [34]. It incorporates the use of normalized results in the evaluation methods to generate standardized risk analysis plots that allows better visualization and easier comparison of resource management policies (explained in Section 4). It also encompasses a more extensive experimental study to better understand the impact of an additional *wait* objective (manage wait time for SLA acceptance) and inaccurate runtime estimates on the achievement of objectives (explained in Section 5 and 6). In particular, experiments are conducted in two possible economic models (commodity market model and bid-based model) to distinguish the performance and volatility of policies across different economic models.

Table I. Focus of four essential objectives.

Focus	Objective	Abbreviation
User-centric	Manage wait time for SLA acceptance	*wait*
	Meet SLA requests	SLA
	Ensure reliability of accepted SLA	*reliability*
Provider-centric	Attain profitability	$profitability$

3. Objectives of a Commercial Computing Service

This section explains why a commercial computing service wants to achieve four essential objectives and how to measure these objectives. As listed in Table I, the four objectives consists of three user-centric objectives: (i) manage wait time for SLA acceptance (*wait*), (ii) meet SLA requests (SLA), (iii) ensure reliability of accepted SLA (*reliability*); and one provider-centric objective: (i) attain profitability ($profitability$).

A user-centric objective can influence service users, whereas a provider-centric objective can only affect computing services. However, in utility computing, a commercial computing service also has to consider or even place greater emphasis on user-centric objectives. This is because a commercial computing service has to be commercially viable and is thus heavily dependent on revenue generated by service users who pay and expect quality-driven and value-added services to be provided.

In addition, we believe that the utility computing model marks an important milestone for the creation of a free market economy to buy and sell computing resources based on actual usage. In this free market economy, we envision the availability of numerous commercial computing services which have the required capability to process any specific job characteristics at any time. These computing services will thus actively compete with one another to increase their market share of service users so as to increase their revenue. This means that service users can switch to any computing service whenever they want. Therefore, ignoring user-centric objectives is likely to result in dwindling number of users, loss of reputation and revenue, and finally out-of-business for a commercial computing service.

We consider all four objectives to be equally important and thus have the same priority as they address various operational aspects of a commercial computing service, from accepting and fulfilling service requests (*wait* and SLA objectives) to monitoring service levels and monetary yields (*reliability* and $profitability$ objectives). However, in the proposed integrated risk analysis (described in Section 4.2), we allow a computing service to prioritize objectives differently by adjusting the corresponding weight of each objective. Hence, this provides the flexibility for different computing services to control the objectives based on their specific interests.

For the measurement of the *wait* objective (in Section 3.1), we decide to compute an average value for it. The average value provides the central tendency of wait times required for various jobs to be accepted. The minimum average value of the *wait* objective is 0 time units.

But, for the measurement of *SLA*, *reliability*, and *profitability* objectives (in Section 3.2, 3.3 and 3.4), we choose to compute a percentage value for each of them. Each percentage value provides a relative performance value with respect to the maximum upper bound value of a specific objective, which is more informative and meaningful, as compared to just having an absolute performance value. As an example, for the *SLA* objective, the total number of m jobs submitted to the computing service is the maximum number of jobs that can possibly have SLA fulfilled by the computing service. Measuring the percentage value of n_{SLA} jobs with SLA fulfilled out of the total number of m jobs submitted is thus more meaningful compared to just the value of n_{SLA}. The minimum and maximum percentage value of *SLA*, *reliability*, and *profitability* objectives is 0% and 100% respectively.

3.1. Manage Wait Time for SLA Acceptance

Customers perceive the responsiveness of a business as a service quality factor because it reflects the willingness of the business to provide fast service and help customers quickly [23]. Moreover, time is a valuable and critical factor for an individual or organization to survive and excel in today's highly dynamic and competitive environment that demands continuous monitoring and quick response. In utility computing, a user submitting the service request for a job has to wait for the commercial computing service to examine and accept the request before starting the actual processing of the job. Thus, we assume that every user accessing the computing service also requires timely processing of their requests in order to satisfy other personal or organizational commitments. In other words, a user who wastes greater time to secure a service request will be more disadvantaged as any delay will impact on the prompt completion of other commitments.

The process of managing the wait time for SLA acceptance can be accessed through the typical amount of time taken by the commercial computing service to accept and execute jobs. Hence, the *wait* objective is measured as:

$$wait = \frac{\sum_{i=1}^{n_{SLA}} tst_i - tsu_i}{n_{SLA}} \tag{1}$$

, where tst_i is the time when job i starts execution after being accepted, tsu_i is the time when job i is submitted to the computing service, and n_{SLA} is the number of jobs with SLA fulfilled. A lower value of $wait$ is better than a higher value.

3.2. Meet SLA Requests

Another service quality factor perceived by customers is the assurance of a business that it is adequately knowledgeable and sufficiently competent [23]. This inspires the trust and confidence of customers in the business. In utility computing, a commercial computing service has to assure service users of its ability to satisfy service demand by meeting SLA requests.

An inability to meet service demand can be verified by a decrease in the number of requested SLAs that are fulfilled successfully. Therefore, the SLA objective is computed as:

$$SLA = \frac{n_{SLA}}{m} * 100 \tag{2}$$

, where n_{SLA} is the number of jobs with SLA fulfilled and m is the number of jobs submitted to the computing service. A higher value of SLA is better than a lower value.

3.3. Ensure Reliability of Accepted SLA

The level of customer satisfaction for a business can be affected by the reliability of the business to deliver expected performance dependably and accurately [23]. In utility computing, since users specify the level of service they require through SLAs, a commercial computing service wants to ensure that it is able to really deliver the agreed level of service by ensuring reliability of accepted SLAs. We assume that a commercial computing service has monitoring mechanisms to check the progress of existing job executions and adjust resources accordingly to meet current and future service obligations.

A compromise in service quality can be ascertained by an increase in the number of accepted SLAs that are not fulfilled successfully. Thus, the *reliability* objective is calculated as:

$$reliability = \frac{n_{SLA}}{n} * 100 \tag{3}$$

, where n_{SLA} is the number of jobs with SLA fulfilled and n is the number of jobs that are accepted by the computing service. A higher value of *reliability* is better than a lower value.

3.4. Attain Profitability

The most important objective for a commercial computing service is to attain profitability as Return On Investment (ROI) for providing the service since commercial businesses are driven by monetary performance and thus need to track their monetary yields. We assume that a commercial computing service has accounting and pricing mechanisms to record resource usage information and compute usage costs to charge service users accordingly.

The cost paid by the service users can also be viewed as the utility or ROI earned by the computing service. Hence, the $profitability$ objective is determined as:

$$profitability = \frac{\sum_{i=1}^{n} u_i}{\sum_{i=1}^{m} b_i} * 100 \tag{4}$$

, where $\sum_{i=1}^{n} u_i$ is the total utility earned from n jobs accepted by the computing service and $\sum_{i=1}^{m} b_i$ is the total budget of m jobs that are submitted to the computing service. A higher value of $profitability$ is better than a lower value.

4. Risk Analysis

A commercial computing service must now be able to assess whether its implemented resource management policy is able to achieve any or all of the objectives to support the utility computing model. This section proposes two evaluation methods that is derived from enterprise risk management [18]: (i) separate and (ii) integrated risk analysis. Both methods evaluate a policy using two indicators: (i) performance and (ii) volatility. Performance acts as the value measure of the policy, while volatility acts as the risk measure. Volatility is selected as the risk measure since it reflects how performance values fluctuate and thus the consistency of the policy in returning similar performance values. This section then describes how the level of associated risk can be easily visualized through risk analysis plots produced from these performance and volatility values.

4.1. Separate Risk Analysis

Separate risk analysis analyzes the performance and volatility involved in a single objective for a particular scenario. An example of a scenario is varying workload whereby only the workload changes while the rest of the experiment settings remains the same. Hence, measuring a specific objective for the varying workload scenario will return a total of n results for each different n workload.

However, the raw values measured for the four objectives do not constitute a consistent and correct outcome. As highlighted in Section 3, a lower value of the *wait* objective is better than a higher value, whereas a higher value of *SLA*, *reliability*, and *profitability* objectives is better than a lower value. Therefore, we normalize these raw values accordingly to obtain normalized values that are standardized within the range 0 to 1, with the minimum value of 0 symbolizing the worst performance and the maximum value of 1 symbolizing the best performance respectively.

The performance μ_{sep} and volatility σ_{sep} for the separate risk analysis of an objective in a particular scenario can be computed as the mean of all n normalized results obtained in the scenario and standard deviation of these n normalized results respectively:

$$performance, \mu_{sep} = \frac{\sum_{i=1}^{n} normalized\_result_i}{n} \tag{5}$$

$$volatility, \sigma_{sep} = \sqrt{\frac{\sum_{i=1}^{n} (normalized\_result_i)^2}{n} - (\mu_{sep})^2} \tag{6}$$

, where $0 \leq normalized\_result_i \leq 1$.

4.2. Integrated Risk Analysis

Since separate risk analysis only examines a single objective and there is more than one objective to realize utility computing, it is critical to be able to assess a combination of multiple objectives in an integrated fashion.

Given that there is a total of n objectives to examine for a particular scenario, the performance μ_{int} and volatility σ_{int} of the integrated risk analysis can be computed using the performance $\mu_{sep,i}$ and volatility $\sigma_{sep,i}$ measures from the separate risk analysis of each objective i:

$$performance, \mu_{int} = \sum_{i=1}^{n} w_i * \mu_{sep,i} \tag{7}$$

$$volatility, \sigma_{int} = \sum_{i=1}^{n} w_i * \sigma_{sep,i} \tag{8}$$

, where $0 \leq w_i \leq 1$ and $\sum_{i=1}^{n} w_i = 1$. w_i is a weight to denote the importance of an objective i with respect to other objectives. These weights for various objectives provide a flexible means for the service provider to easily adjust the importance of an objective and determine

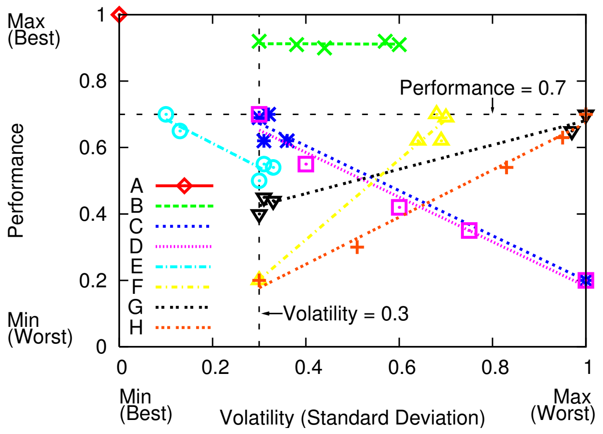


Figure 1. Sample risk analysis plot of policies.

Table II. Performance and volatility of policies in the sample risk analysis plot.

Policy	Performance			Volatility		
	Maximum	Minimum	Difference	Maximum	Minimum	Difference
A	1.0	1.0	0.0	0.0	0.0	0.0
B	0.9	0.9	0.0	0.6	0.3	0.3
C	0.7	0.2	0.5	1.0	0.3	0.7
D	0.7	0.2	0.5	1.0	0.3	0.7
E	0.7	0.5	0.2	0.3	0.1	0.2
F	0.7	0.2	0.5	0.7	0.3	0.4
G	0.7	0.4	0.3	1.0	0.3	0.7
H	0.7	0.2	0.5	1.0	0.3	0.7

its level of impact on the overall achievement of a combination of objectives. For the experiments, since we consider all the objectives to be of equal importance, w_i of each objective i in a combination of three objectives and all four objectives are thus 0.33 (1/3) and 0.25 (1/4) respectively.

4.3. Risk Analysis Plot

Risk analysis plots can now be generated using the performance and volatility values of various resource management policies in all the scenarios. A risk analysis plot can be generated for a single objective (using separate risk analysis) or a combination of objectives (using integrated risk analysis) to easily visualize the level of associated risk for achieving them. Figure 1 shows a sample risk analysis plot of eight policies for five scenarios.

Table III. Ranking of policies based on best performance in the sample risk analysis plot.

Rank	Policy	Maximum performance	Minimum volatility	Performance difference	Volatility difference	Gradient of trend line
1	A	1.0	0.0	0.0	0.0	NA
2	B	0.9	0.3	0.0	0.3	Zero
4	E	0.7	0.1	0.2	0.2	Decreasing
3	G	0.7	0.3	0.3	0.7	Increasing
5	F	0.7	0.3	0.5	0.4	Increasing
6	C	0.7	0.3	0.5	0.7	Decreasing
7	D	0.7	0.3	0.5	0.7	Decreasing
8	H	0.7	0.3	0.5	0.7	Increasing

Table IV. Ranking of policies based on best volatility in the sample risk analysis plot.

Rank	Policy	Minimum volatility	Maximum performance	Volatility difference	Performance difference	Gradient of trend line
1	A	0.0	1.0	0.0	0.0	NA
2	E	0.1	0.7	0.2	0.2	Decreasing
3	B	0.3	0.9	0.3	0.0	Zero
4	F	0.3	0.7	0.4	0.5	Increasing
5	G	0.3	0.7	0.7	0.3	Increasing
6	C	0.3	0.7	0.7	0.5	Decreasing
7	D	0.3	0.7	0.7	0.5	Decreasing
8	H	0.3	0.7	0.7	0.5	Increasing

In a risk analysis plot, each point for a policy represents the performance and volatility of that policy for a particular scenario. Since the sample plot in Figure 1 considers five scenarios, there can only be a maximum of five different points for a policy in the plot. A trend line may then be plotted using these points to reflect the general performance and volatility of that policy. A policy cannot have a trend line if it does not have any or too few different points for all the various scenarios. For example, in Figure 1, policy A has the same points for all scenarios and thus does not have a trend line.

Table II shows the maximum and minimum values of the eight policies for performance and volatility in Figure 1 and their respective differences. A policy achieving a higher performance is better than achieving a lower performance, whereas a policy achieving a higher volatility is worse than achieving a lower volatility. This is because a higher volatility means that performance results fluctuate more, thus increasing the possibility that the same performance cannot be achieved for various scenarios. In a risk analysis plot, the best performance that can be achieved by a policy is the maximum performance value of 1. On the other hand, the best volatility that can be achieved by a policy is the minimum volatility value of 0. Therefore, in Figure 1, policy A is the best ideal policy since it achieves the same best ideal performance of 1 for all five scenarios, and thus also the best ideal volatility of 0.

Table III and IV shows the ranking of policies based on best performance and volatility respectively in the sample risk analysis plot. For best performance, a policy is ranked in the following order: (i) maximum performance, (ii) minimum volatility, (iii) performance difference, (iv) volatility difference, and (v) gradient of trend line. For best volatility, the volatility of a policy is first considered before its performance. Hence, for best volatility, a policy is ranked in the following order: (i) minimum volatility, (ii) maximum performance, (iii) volatility difference, (iv) performance difference, and (v) gradient of trend line.

A higher value is preferred for maximum performance, but a lower value is preferred for minimum volatility. For both performance and volatility differences, a lower value is preferred as it represents a shorter range of possibilities. The preferred order of gradient is as follows: (i) decreasing, (ii) increasing, and (iii) zero. A decreasing gradient indicates a lower volatility for higher performance, whereas an increasing gradient indicates a higher volatility. A zero gradient signifies changing volatility with no change in performance. Thus, in Table III and IV, policies C and D are better than policy H as they have decreasing gradients. But, policy C is better than policy D because most of the points (four of five points) for policy C are near to its maximum performance of 0.7 and minimum volatility of 0.3, compared to the evenly distributed points for policy D.

5. Performance Evaluation

In order to thoroughly demonstrate the applicability of separate and integrated risk analysis, this section investigates whether a commercial computing service is able to achieve the objectives in two possible economic models: (i) commodity market model and (ii) bid-based model. This section first describes the differences between the commodity market model and bid-based model, before specifying the various resource management policies that will be examined for each of them. It then ex-

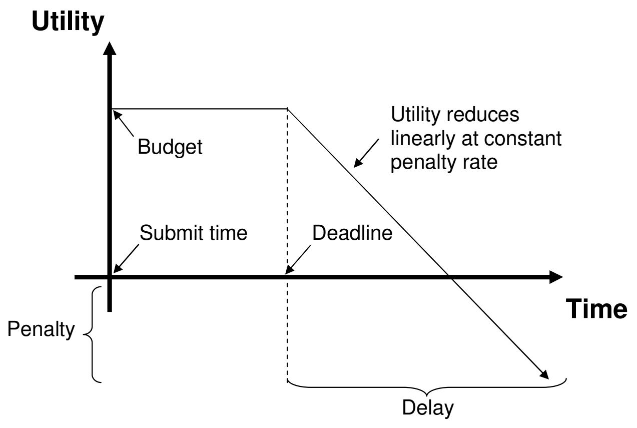


Figure 2. Bid-based model: Impact of penalty function on utility.

plains the evaluation methodology, followed by outlining the scenarios for the experiments.

5.1. Economic Models

In this chapter, there are two differences between the commodity market model and bid-based model. The first difference is how to determine the price. In the commodity market model, the commercial computing service specifies the price that users will pay for the amount of resources consumed. Pricing parameters can be usage time and usage quantity, while prices can be flat or variable. A flat price means that pricing is fixed for a certain time period, whereas a variable price means that pricing changes over time. However, the commercial computing service can only charge a cost which is lower than or equal to the maximum budget specified by the user when he submits the job. This also means that a job will be rejected by the commercial computing service if the expected cost of the job is higher than the specified budget. In the bid-based model, the user provides the bid or price that he will pay the commercial computing service for completing the job.

The second difference is any penalty involved when the commercial computing service fails to meet a SLA (which is to complete a job within its deadline). In the commodity market model, there is no penalty involved. The commercial computing service continues to charge the user based on the usual pricing parameter and price. But in the bid-based model, the commercial computing service is liable to be penalized based on the penalty function shown in Figure 2. The penalty function penalizes the commercial computing service by reducing the budget of a job over time after the lapse of its deadline. For simplicity, we model the penalty function as linear, as in other previous works [5][12][22]. For every job i, the commercial computing service earns a utility u_i

Table V. Policies for performance evaluation.

Policy	Economic model		Primary scheduling parameter			
	Commodity market model	Bid-based model	Arrival time	Runtime	Deadline	Budget with penalty
FCFS-BF	✓	✓	✠			
SJF-BF	✓			✠		
EDF-BF	✓	✓			✠	
Libra	✓	✓			✠	
Libra+$	✓				✠	
LibraRiskD		✓			✠	
FirstReward		✓				✠

depending on its penalty rate pr_i and delay dy_i:

$$u_i = b_i - (dy_i * pr_i) \tag{9}$$

Job i has a delay dy_i if it needs a longer time to complete than its specified deadline d_i:

$$dy_i = (tf_i - tsu_i) - d_i \tag{10}$$

where tsu_i is the time when job i is submitted into the computing service and tf_i is the time when job i is finished. Thus, job i has no delay (i.e. $dy_i = 0$) if it finishes before the deadline and the computing service earns the full budget b_i as utility u_i. But, if there is a delay (i.e. $dy_i > 0$), u_i drops linearly until it turns negative and becomes a penalty (i.e. $u_i < 0$). As shown in Figure 2, the penalty is unbounded till the time when the job is finally completed. This implies that the commercial computing service must be cautious about accepting new jobs to ensure that not too many jobs are accepted such that heavily penalized jobs dramatically erode previously earned utility.

5.2. Resource Management Policies.

Table V lists five resource management policies to be examined for each economic model and the primary scheduling parameter they consider to allocate resources to jobs. This subsection first describes how each policy works in general, before explaining the difference between policies examined in the commodity market model and bid-based model.

FCFS-BF, SJF-BF, and EDF-BF are backfilling policies which prioritize jobs based on arrival time (First Come First Serve), runtime (Shortest Job First), and deadline (Earliest Deadline First) respectively. All three policies adopt EASY backfilling [17][19] to increase resource utilization. A queue is used to store incoming jobs as only a single job can run at a processor at any time (i.e. space-shared). When insufficient number of processors is available for the first job (with the highest priority) in the queue, EASY backfilling assigns these unused processors to the next waiting jobs in the queue provided that they do not delay the first job based their runtime estimates. In other words, jobs that skip ahead must finish before the time when the required number of processors by the first job is expected to be available.

These three variations of EASY backfilling policy are chosen for comparison because EASY backfilling is currently the most widely used policy for scheduling parallel jobs in commercial cluster batch schedulers [7]. However, we find that these policies without job admission control perform much worse, especially when deadlines of jobs are short. Hence, we implement a generous admission control that checks whether a job should be rejected based on two conditions before running it: (i) the job is predicted to exceed its deadline based on its runtime estimate, and (ii) the job has already exceeded its deadline while waiting in the queue. This generous admission control enables FCFS-BF, SJF-BF, and EDF-BF to select their highest priority job at the latest time, while ensuring that earlier jobs whose deadlines have lapsed do not incur propagated delay for later jobs.

Libra [24] uses deadline-based proportional processor share with job admission control to enforce the deadlines of jobs. A minimum processor time share is computed for each job i as tr_i/d_i using its runtime estimate tr_i and deadline d_i so that job i is accepted only if there are sufficient required number of processors with the free minimum processor time share. This means that multiple jobs can run at a processor at any time, using its allocated minimum processor time share (i.e. time-shared). Unlike the above backfilling policies, no queue is maintained so a new job is checked during submission and rejected immediately if its deadline is not expected to be fulfilled. Libra chooses suitable processors based on the best fit strategy, i.e. processors that have the least available processor time left with the new job will be selected first so that every processor is saturated to its maximum. Any remaining free processor time is then distributed among all jobs at the processor according to the computed processor time share of each job.

Libra+\$ [35] is Libra with an enhanced pricing function that satisfies four essential requirements for pricing of resources to prevent workload overload: (i) flexible, (ii) fair, (iii) dynamic, and (iv) adaptive. The price P_{ij} for per unit of resource utilized by job i at compute node j is computed as: $P_{ij} = (\alpha * PBase_j) + (\beta * PUtil_{ij})$. The base price $PBase_j$ is a static pricing component for utilizing a resource at node j. The utilization price $PUtil_{ij}$ is a dynamic pricing component which is computed as a factor of $PBase_j$ based on the utilization of the resource at node j for the required deadline of job i: $PUtil_{ij} = RESMax_j / RESFree_{ij} * PBase_j$. $RESMax_j$ and $RESFree_{ij}$ are the maximum units and remaining free units of the resource at node j for the deadline duration of job i respectively. Hence, $RESFree_{ij}$ has been deducted units of resource committed for other confirmed reservations and job i for its deadline duration.

The factors α and β for the static and dynamic components of Libra+$ respectively provides the flexibility for the cluster owner to easily configure and modify the weightage of the static and dynamic components on the overall price P_{ij}. Libra+$ is fair since jobs are priced based on the amount of different resources utilized. It is also dynamic because the overall price of a job varies depending on the availability of resources for the required deadline. Finally, it is adaptive as the overall price is adjusted depending on the current supply and demand of resources to either encourage or discourage job submission. For the experiments, α is 1 and β is 0.3.

LibraRiskD [33] is also an improvement of Libra and uses the same deadline-based proportional processor share. The difference is that LibraRiskD considers the risk of deadline delay when selecting suitable nodes for a new job. Nodes are selected for a new job only if they have zero risk of deadline delay. This enables LibraRiskD to manage the risk of inaccurate runtime estimates more effectively than Libra. LibraRiskD is thus able to complete more jobs with deadline fulfilled and achieve lower average slowdown than Libra.

FirstReward [12] determines possible future earnings PV_i with possible opportunity cost penalties $cost_i$ based on the penalty rate pr_i and estimated remaining runtime RPT_i of a job i. The reward $reward_i$ is then calculated through a α-weighting function as: $reward_i = ((\alpha * PV_i) - ((1-\alpha) * cost_i))/RPT_i$. The earnings PV_i of a job i is computed as: $PV_i = b_i/(1 + (discount\_rate * RPT_i))$, where b_i is the budget of job i. For unbounded penalties, the penalty cost $cost_i$ of a job i is the sum of penalty for all other n accepted jobs based on RPT_i: $cost_i = \sum_{j=0;j\neq i}^{n} (pr_j * RPT_i)$. The admission control of FirstReward computes the slack $slack_i$ of a new job i during submission and rejects the job immediately if $slack_i$ is less than a specified slack threshold: $slack_i = (PV_i - cost_i)/pr_i$. The slack threshold determines the balance of earnings and penalties where a high threshold avoids future commitments that can result in possible penalties. Setting the correct slack threshold is not trivial as the ideal slack threshold changes depending on the workload. After testing various slack threshold values for the simulated workload, we derive the following ideal simulation settings for FirstReward: α is 1, the discount rate is 1%, and the slack threshold is 25. We have also extended the FirstReward to consider multiple-processor parallel jobs since the original work only considers single-processor jobs. However, we do not make FirstReward to support backfilling, so delays may occur due to waiting for the required number of processors.

Table V shows that some policies (FCFS-BF, EDF-BF, and Libra) are examined in both commodity market model and bid-based model. As previously explained in Section 5.1, the only difference between these two models is how to determine the utility earned by the commercial computing service.

In the commodity market model, the policies (FCFS-BF, SJF-BF, EDF-BF, Libra, and Libra+$) earn utility based on the different pricing rates each of them charge. However, the maximum utility that can be earned for a job is restricted by the user-specified budget. In other words, a job that is expected to cost more than the specified budget will be rejected. FCFS-BF, SJF-BF, and EDF-BF charge the user based on the static base price $PBase_j$ of using processing time at node j, so the commercial computing service earns a utility of $tr_i * PBase_j$ for job i with runtime tr_i. For the experiments, $PBase_j$ is \$1 per second for all nodes. Libra uses a static pricing function that provides incentives for jobs with a more relaxed deadline to compute a utility of $\gamma * tr_i + \delta * tr_i / d_i$ for job i with runtime tr_i and deadline 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 a more relaxed deadline, so as to encourage users to submit jobs with longer deadlines. For the experiments, both γ and δ are 1. Libra+$ uses an enhanced pricing function as described earlier in this section. But, for a job i, Libra+$ can compute different price P_{ij} at each node j since workload conditions can vary at different nodes. Hence, to maximize revenue, Libra+$ uses the highest price P_{ij} among allocated nodes as the price for job i.

On the other hand, in the bid-based model, all policies (FCFS-BF, EDF-BF, Libra, LibraRiskD, and FirstReward) can earn a maximum utility equal to the budget (bid) of the job specified by the user for completing the job within its deadline. If these policies cannot complete a job within its deadline, the utility reduces and can instead become a penalty depending on when the job is eventually completed (as described in Section 5.1).

Finally, all the policies are assumed to be non-preemptive. In other words, jobs that are started need to complete entirely and are not paused or terminated after the lapse of their deadlines. This leads to the issue of whether the non-preemptive policies will be affected by the inaccuracy of runtime estimates.

Under estimation of runtime estimates of previously accepted jobs can result in delays that cause later accepted jobs not to finish within their expected deadlines. For the commodity market model, potential utility is lost when fewer later arriving jobs are accepted due to the delays caused by previously accepted jobs. For the bid-based model, the loss of utility can be caused by accepting fewer later arriving jobs and paying penalties to compensate users for delays.

Conversely, over estimation of runtime estimates allows fewer jobs to be accepted since admission controls unnecessarily reject jobs after predicting that their deadlines cannot be fulfilled. For the commodity market model, higher utility may however be gained as the prices charged are computed using the over-estimated runtime estimates. But, for the bid-based model, potential utility is lost as fewer jobs are accepted.

5.3. Evaluation Methodology

The evaluation uses a discrete event simulator called GridSim [4][25] to run the experiments. The experiments are generated from 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 [8].

The SDSC SP2 trace is chosen because it has the highest resource utilization of 83.2% among other traces to ideally model the heavy workload scenario for a computing service. This 5000 job subset based on the last 3.75 months of the SDSC SP2 trace requires an average of 17 processors and has an average inter arrival time of 1969 seconds (32.8 minutes) and average runtime of 8671 seconds (2.4 hours). The computing service that is simulated resembles the IBM SP2 at San Diego Supercomputer Center (SDSC) with 128 compute nodes, each having a SPEC rating of 168.

However, jobs submitted to a commercial computing service in utility computing need to have three other significant parameters (deadline, budget, and penalty) which is unfortunately unavailable in this trace and from an actual commercial computing service. Therefore, we adopt a similar methodology in [12] to model these parameters through two job classes: (i) high urgency and (ii) low urgency.

Given that a job i has deadline d_i, budget b_i, penalty rate pr_i, and runtime tr_i, jobs in the *high urgency* class has a deadline of low d_i/tr_i value, budget of high $b_i/f(tr_i)$ value, and penalty rate of high $pr_i/g(tr_i)$ value. $f(tr_i)$ and $g(tr_i)$ are functions to represent the minimum budget and penalty rate that the user will quote with respect to runtime tr_i.

Conversely, each job in the *low urgency* class has a deadline of high d_i/tr_i value, budget of low $b_i/f(tr_i)$ value, and penalty rate of low $pr_i/g(tr_i)$ 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 and also specify a higher penalty if the job is delayed beyond its deadline. The arrival sequence of jobs from the high urgency and low urgency classes is randomly distributed.

Values are normally distributed within each of the three parameters. The ratio of each parameter's high-value mean and low-value mean is thus known as the *high:low ratio*. A higher deadline high:low ratio indicates that low urgency jobs have longer deadlines than that of a lower ratio. For instance, a deadline high:low ratio of 8 means the d_i/tr_i mean of low urgency jobs is two times more than that of a deadline high:low ratio of 4. On the other hand, a higher budget or penalty high:low ratio denotes that high urgency jobs have larger budget or penalty than that of a lower ratio.

Since the deadline, budget and penalty rate of a job will now always be set as a larger factor of runtime, we introduce a *bias* parameter to counter against this issue. For example, the deadline bias bd_i works such that a job i with a runtime more than the average runtime of all jobs (i.e. longer runtime) will have a deadline $d_i = d_i/bd_i$ (i.e. shorter deadline). But if job i has a runtime less than the average runtime of all jobs (i.e. shorter runtime), then it will have $d_i = d_i * bd_i$ (i.e. longer deadline). This works likewise for budget and penalty bias.

Different levels of workload are modeled through 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 arrival delay factor represents higher workload by shortening the inter arrival time of jobs.

The *inaccuracy* of runtime estimates is measured with respect to the actual runtime estimates of jobs obtained from the trace. An inaccuracy of 100% is equivalent to the actual runtime estimates from the trace, whereas an inaccuracy of 0% assumes runtime estimates are accurate and equal to the real runtimes of the jobs. For the actual runtime estimates from the last 5000 job subset of the SDSC SP2 trace, only 8% of them are under estimates, while the remaining 92% of them are over estimates. This means that runtime estimates provided by users are often over estimated.

Table VI. Varying values of twelve scenarios.

Percentage of high urgency jobs	Arrival delay factor	Percentage of inaccuracy of runtime estimates	Bias (deadline, budget, penalty)	High:low ratio (deadline, budget, penalty)	Low-value mean (deadline, budget, penalty)
0	0.02	(Set A) <u>0</u>	<u>1</u>	1	1
<u>20</u>	0.10	20	2	2	2
40	0.25	40	4	<u>4</u>	<u>4</u>
60	0.50	60	6	6	6
80	0.75	80	8	8	8
100	<u>1.00</u>	(Set B) <u>100</u>	10	10	10

Note: <u>underline</u> denotes default value.

5.4. Scenarios

We can now apply separate and integrated risk analysis (introduced in Section 4) to assess the resource management policies with respect to the four essential objectives (defined in Section 3). For each objective, we consider twelve scenarios. Table VI lists the twelve scenarios and their varying values for the experiments.

Each of the varying bias, varying high:low ratio, and varying low-value mean scenarios is for the deadline, budget, and penalty parameters respectively, thus creating a total of nine scenarios. For simplicity, we have set the deadline, budget, and penalty parameters to have the same default and varying values for varying bias, varying high:low ratio, and varying low-value mean scenarios. The three other remaining scenarios are varying percentage of high urgency jobs (job mix), varying arrival delay factor (workload), and varying percentage of inaccuracy of runtime estimates. For each scenario, there is only one set of varying values, while the rest of the experiment settings remains the same with default values (underlined in Table VI). Table VI also shows six varying values in each scenario, thus deriving six normalized results to compute the separate risk analysis of a particular objective.

Previous studies [19][28] have shown that runtime estimates provided by users are rather inaccurate. Hence, for each economic model (commodity market model and bid-based model), we run two different sets of experiments: (i) Set A and (ii) Set B to examine the impact of inaccuracy of runtime estimates on the achievement of objectives. The only different setting between Set A and Set B is the default value for percentage of inaccuracy of runtime estimates as shown in Table VI: Set A has 0% of inaccuracy to represent accurate runtime estimates, while

Set B has 100% of inaccuracy to represent actual runtime estimates from the trace.

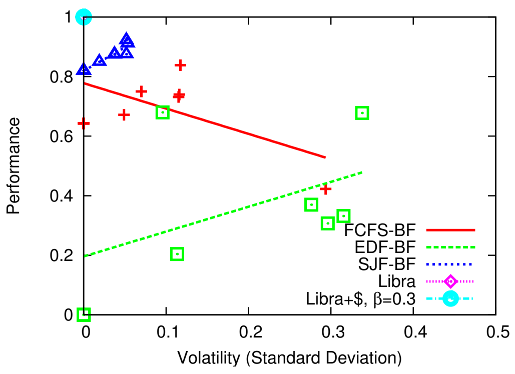
6. Performance Results

This section analyzes the performance results of various resource management policies for two economic models: (i) commodity market model and (ii) bid-based model. As there are four essential objectives to realize utility computing (as defined in Section 3), it examines the performance results using three approaches for each economic model: (i) separate risk analysis of one objective, (ii) integrated risk analysis of three objectives, and (iii) integrated risk analysis of all four objectives. The integrated risk analysis of three objectives enables the understanding of how the combination of all the other remaining objectives will perform in the absence of a particular objective.

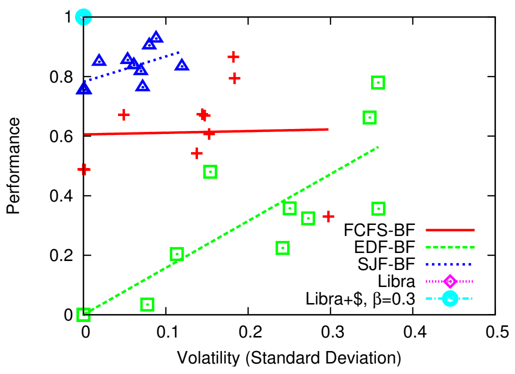
6.1. Commodity Market Model

Figure 3 shows the separate risk analysis of one objective (*wait*, *SLA*, *reliability*, and *profitability*) in Set A (accurate runtime estimates) and Set B (actual runtime estimates from the trace) for the commodity market model. For the *wait* objective in Set A (Figure 3a) and Set B (Figure 3b), Libra and Libra+$ have the best ideal performance and volatility of 1 and 0 respectively since jobs are examined immediately after submission to determine whether their deadlines can be fulfilled or not. On the other hand, FCFS-BF, SJF-BF, and EDF-BF have lower performance and higher volatility as they keep jobs in queues and examine them only prior to execution to enable a better selection choice. Out of these three policies, SJF-BF returns the best performance and volatility because it selects the shortest job to execute first and thus requires queued jobs to wait the least before being examined. EDF-BF returns the worst performance and volatility since jobs that arrive later but have earlier deadlines will execute first, thus delaying other jobs submitted earlier before them.

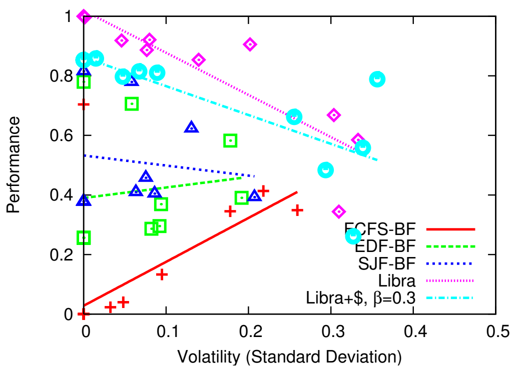
For the *SLA* objective in Set A (Figure 3c) and Set B (Figure 3d), SJF-BF has either similar or better performance than EDF-BF, while EDF-BF has better performance than FCFS-BF. This is due to the fact that the evaluation methodology always set the deadline of a job as a larger factor of its runtime for all scenarios (except deadline bias). Thus, SJF-BF has the best performance among these three policies by executing the job with the shortest runtime first, whereas FCFS-BF has the worst performance by considering the arrival time and not deadline.



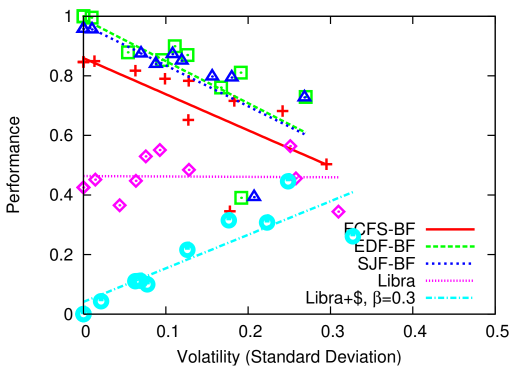
a. Set A: *wait*



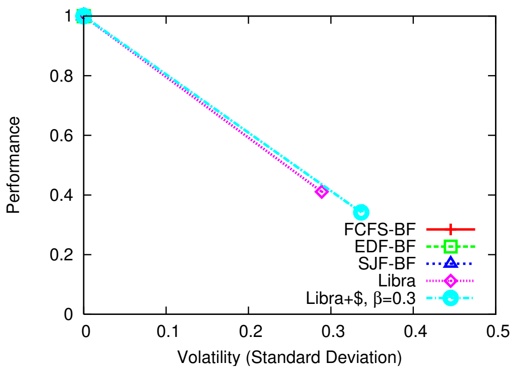
b. Set B: *wait*



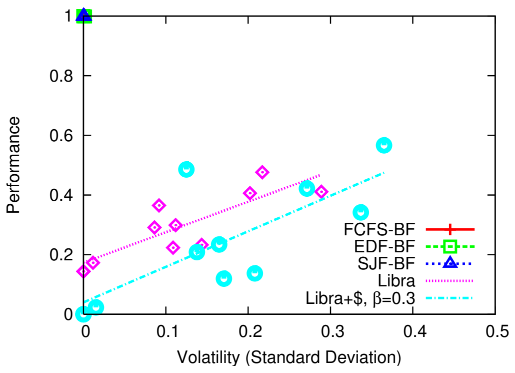
c. Set A: *SLA*



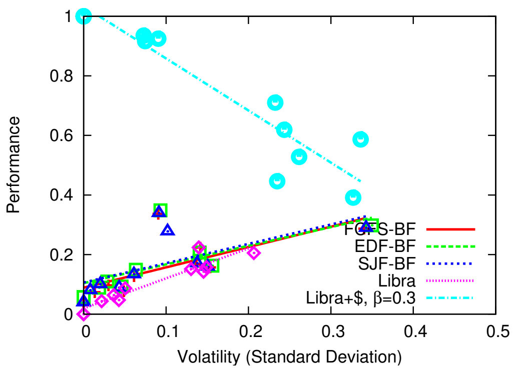
d. Set B: *SLA*



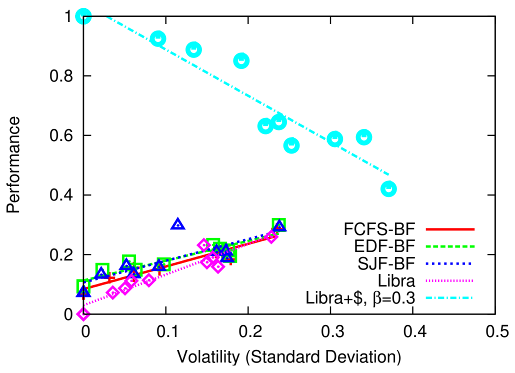
e. Set A: *reliability*



f. Set B: *reliability*



g. Set A: *profitability*



h. Set B: *profitability*

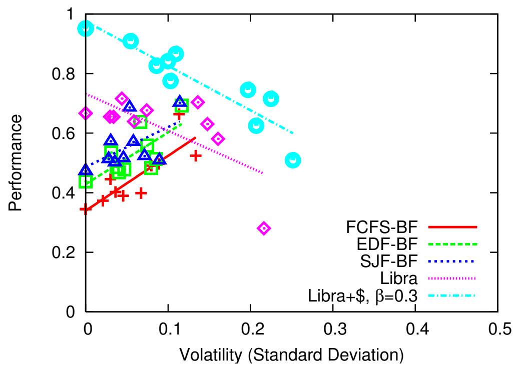
Figure 3. Commodity market model: Separate risk analysis of one objective

Libra+$ has lower performance and slightly higher volatility than Libra in Set A and Set B since it accepts and fulfills a lower number of jobs by increasing the pricing as the workload increases. In Set A, Libra and Libra+$ have the best performance. But, in Set B, Libra and Libra+$ have the worst performance for the similar volatility as that of FCFS-BF, SJF-BF, and EDF-BF. In particular, Libra and Libra+$ have increasing performance with decreasing volatility (decreasing gradient) in Set A which is a better result, compared to constant or increasing performance with increasing volatility (zero and increasing gradient) in Set B which is a worse result.

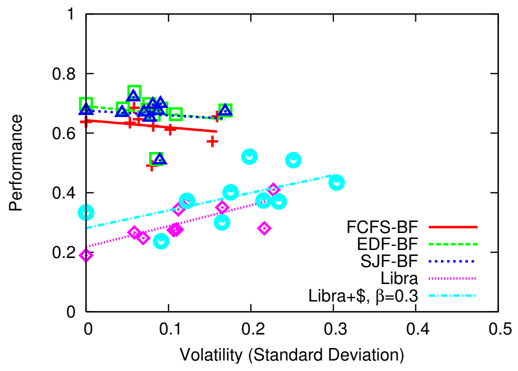
This highlights the issue of inaccurate runtime estimates. Libra and Libra+$ assume accurate runtime estimates and thus accept a lower number of jobs in Set B with the actual runtime estimates from the trace being inaccurate. FCFS-BF, SJF-BF, and EDF-BF are less affected than Libra and Libra+$ in Set B because of the generous admission control we implemented for them. New jobs are only examined and accepted prior to execution when the previously accepted jobs are completed and not during job submission in the case of Libra and Libra+$.

For the *reliability* objective in Set A (Figure 3e) and Set B (Figure 3f), the impact of inaccurate runtime estimates on Libra and Libra+$ can be clearly seen. In Set A, Libra and Libra+$ have a single point deviation due to the scenario of inaccuracy of runtime estimates. In Set B, Libra and Libra+$ have substantially lower performance and higher volatility as the actual runtime estimates from the trace are highly inaccurate. In contrast, FCFS-BF, SJF-BF, and EDF-BF have the best ideal performance and volatility of 1 and 0 respectively in Set A and Set B.

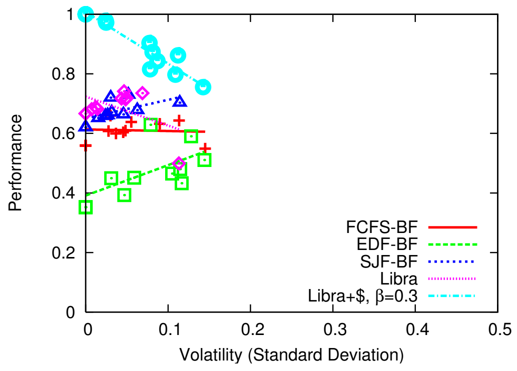
For the $profitability$ objective in Set A (Figure 3g) and Set B (Figure 3h), Libra+$ has the best performance. In addition, only Libra+$ has increasing performance with decreasing volatility (decreasing gradient), whereas all other policies have increasing performance with increasing volatility (increasing gradient). This demonstrates the effectiveness of the enhanced pricing function used by Libra+$ to gain significantly higher utility than all other policies, even when the number of jobs accepted is lower in Set B (Figure 3d). However, Libra+$ has higher volatility than all other policies in Set B.



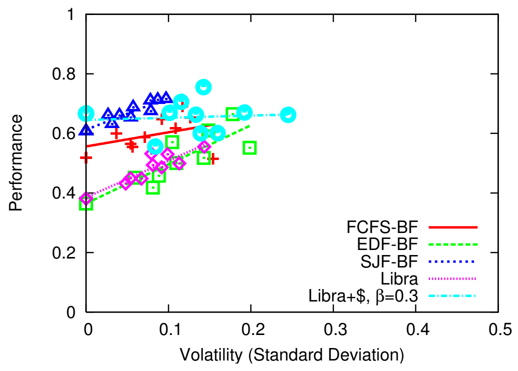
a. Set A: *SLA*, *reliability*, *profitability*



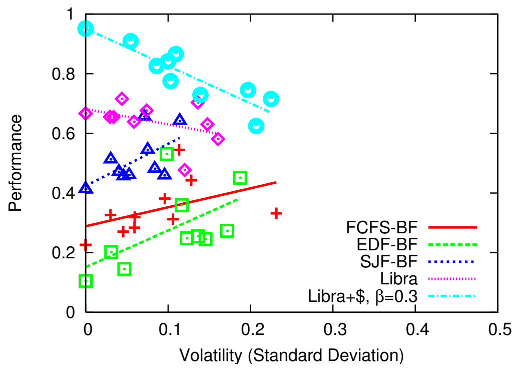
b. Set B: *SLA*, *reliability*, *profitability*



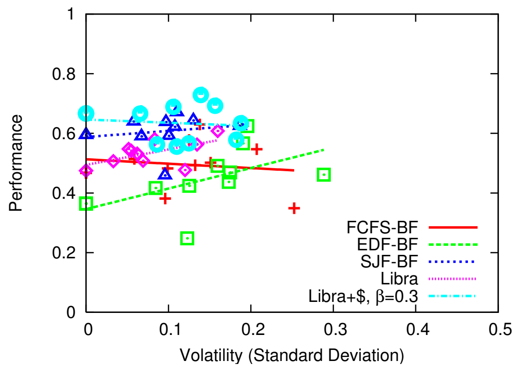
c. Set A: *wait*, *reliability*, *profitability*



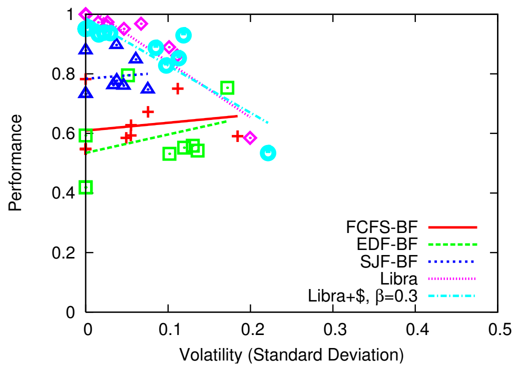
d. Set B: *wait*, *reliability*, *profitability*



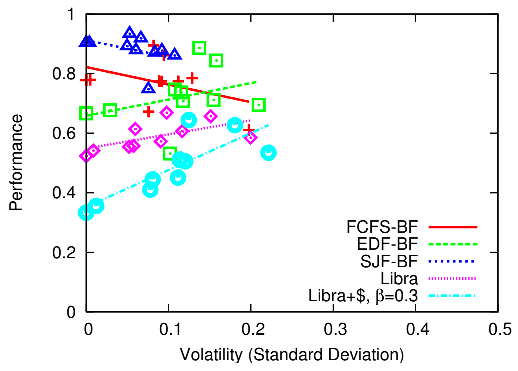
e. Set A: *wait*, *SLA*, *profitability*



f. Set B: *wait*, *SLA*, *profitability*

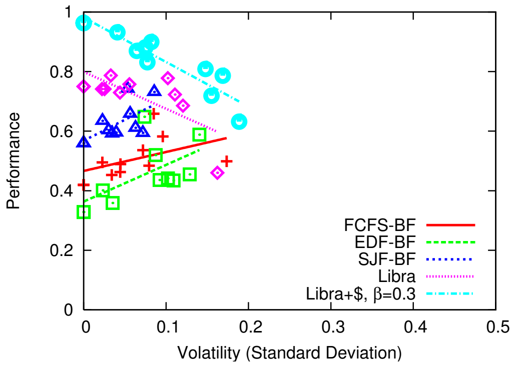


g. Set A: *wait*, *SLA*, *reliability*

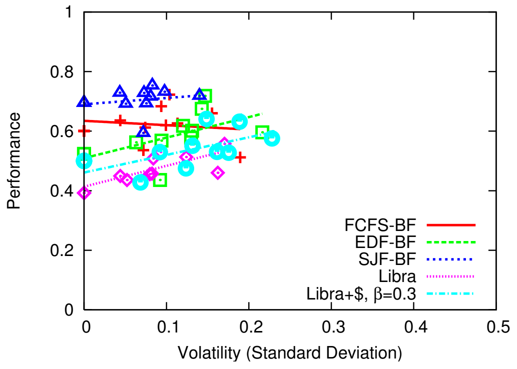


h. Set B: *wait*, *SLA*, *reliability*

Figure 4. Commodity market model: Integrated risk analysis of three objectives



a. Set A: *wait*, *SLA*, *reliability*, *profitability*



b. Set B: *wait*, *SLA*, *reliability*, *profitability*

Figure 5. Commodity market model: Integrated risk analysis of all four objectives

Figure 4 shows the integrated risk analysis of three objectives in Set A (accurate runtime estimates) and Set B (actual runtime estimates from the trace) for the commodity market model. For the three combinations of objectives that include the *profitability* objective in Set A (Figure 4a, 4c, and 4e) and Set B (Figure 4b, 4d, and 4f), Libra+$ has higher performance than Libra. Libra+$ has lower performance than Libra only for the combination of objectives without the *profitability* objective in Set A (Figure 4g) and Set B (Figure 4h). This reinforces that Libra+$ is able to gain higher utility through its enhanced pricing function.

Again, the inaccuracy of runtime estimates can be observed to dramatically affect the performance of Libra and Libra+$. For the combination of objectives without *SLA* (Figure 4d) and *reliability* (Figure 4f) objectives, the performance of Libra and Libra+$ are somewhat similar or slightly worse than FCFS-BF, SJF-BF, and EDF-BF. But, for the combination of objectives without *wait* (Figure 4b) and *profitability* (Figure 4h) objectives, the performance of Libra and Libra+$ are much worse than FCFS-BF, SJF-BF, and EDF-BF. Libra and Libra+$ are thus able to achieve considerably better performance than FCFS-BF, SJF-BF, and EDF-BF for *wait* and *profitability* objectives.

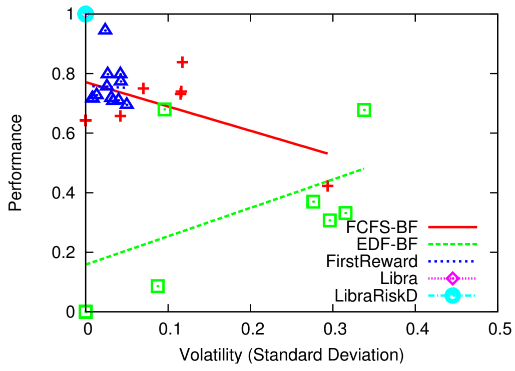
Another interesting observation pertains to the three backfilling policies: FCFS-BF, SJF-BF, and EDF-BF. For the three combinations of objectives that include the *wait* objective in Set A (Figure 4c, 4e, and 4g) and Set B (Figure 4d, 4f, and 4h), SJF-BF has the best performance and volatility, while EDF-BF has the worst performance. When the *wait* objective is excluded in Set A (Figure 4a) and Set B (Figure 4b), these three policies have almost similar performance and volatility. This highlights that the amount of wait time for SLA acceptance incurred by these three policies for the *wait* objective critically affects the overall achievement of objectives.

Figure 5 shows the integrated risk analysis of all four objectives in Set A (accurate runtime estimates) and Set B (actual runtime estimates from the trace) for the commodity market model. In Set A (Figure 5a), Libra and Libra+$ have the best performance. In particular, Libra and Libra+$ have increasing performance with decreasing volatility (decreasing gradient) which is better, compared to FCFS-BF, SJF-BF, and EDF-BF which have increasing performance with increasing volatility (increasing gradient). Libra+$ is able to achieve the best performance due to the capability of its enhanced pricing function to achieve very much better performance for the $profitability$ objective.

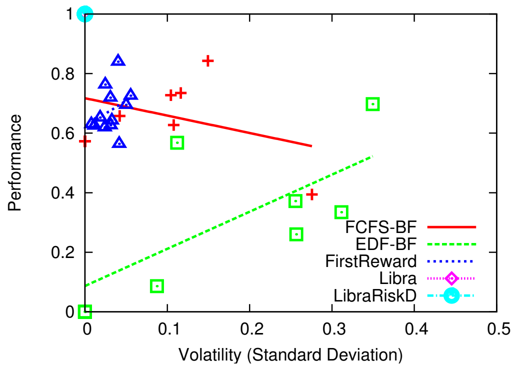
But, in Set B (Figure 5b), Libra and Libra+$ have the worst performance as the actual runtime estimates from the trace are inaccurate. Libra and Libra+$ also have increasing performance with increasing volatility (increasing gradient) in Set B which is worse, compared to having increasing performance with decreasing volatility (decreasing gradient) in Set A. Instead, SJF-BF has the best performance and volatility in Set B. This exposes the weakness of non-preemptive policies using admission controls that rely on accurate runtime estimates to examine and accept jobs at job submission, especially when the actual runtime estimates from the trace are highly inaccurate.

6.2. Bid-based Model

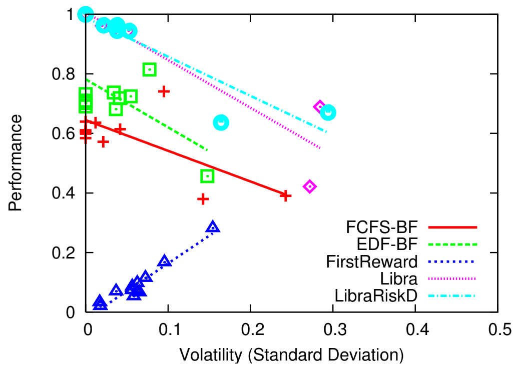
Figure 6 shows the separate risk analysis of one objective (*wait*, SLA, *reliability*, and $profitability$) in Set A (accurate runtime estimates) and Set B (actual runtime estimates from the trace) for the bid-based model. For the *wait* objective in Set A (Figure 6a) and Set B (Figure 6b), Libra and LibraRiskD have the best ideal performance and volatility of 1 and 0 respectively since jobs are examined immediately after submission to determine whether their deadlines can be fulfilled or not. FirstReward has the next best performance and volatility as it also examines new jobs immediately after submission. But, FirstReward has lower performance and higher volatility than Libra and LibraRiskD because of two reasons. The first reason is that FirstReward delays previously accepted jobs to accept new jobs that are more profitable. The second reason is that FirstReward is unable to start the execution of accepted jobs immediately if the required number of processors is not available due to its assumption of space-shared execution. On the other hand, Libra and LibraRiskD assume time-shared execution and immediately starts the execution of accepted jobs by allocating processor time based on the deadline and runtime estimate of each job.



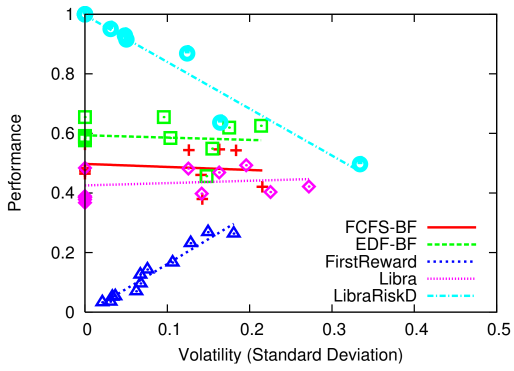
a. Set A: *wait*



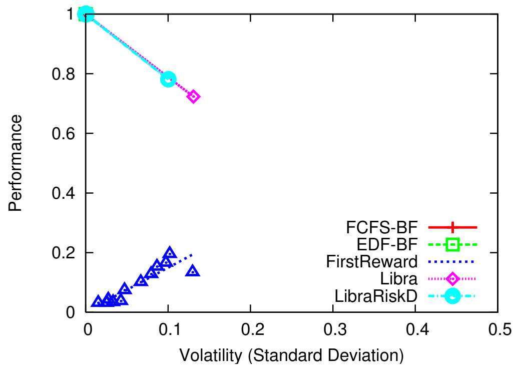
b. Set B: *wait*



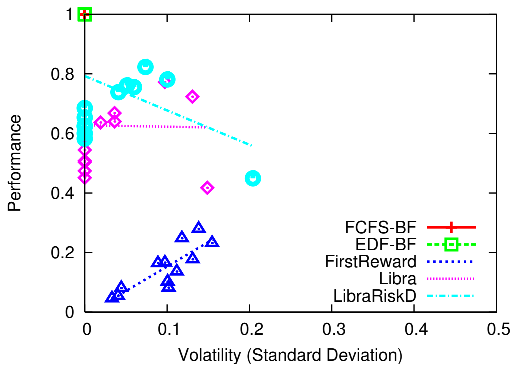
c. Set A: *SLA*



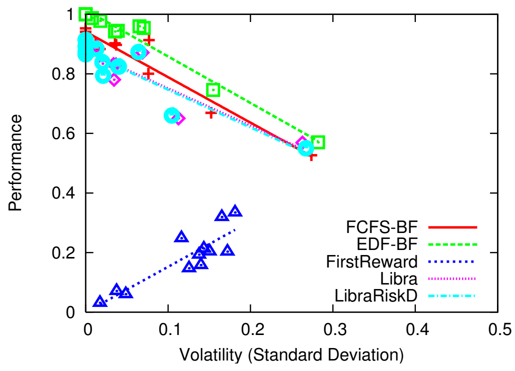
d. Set B: *SLA*



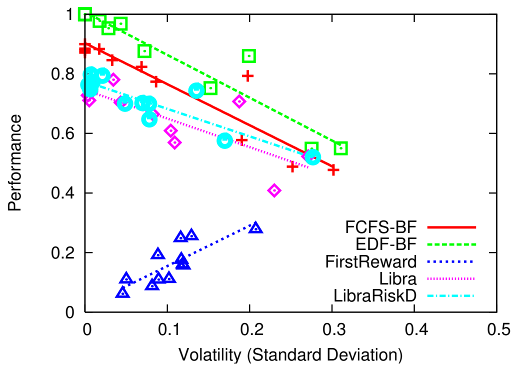
e. Set A: *reliability*



f. Set B: *reliability*



g. Set A: *profitability*



h. Set B: *profitability*

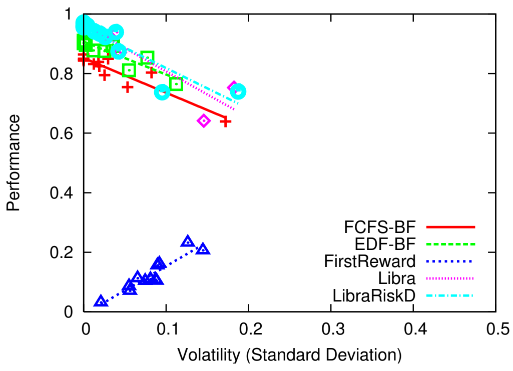
Figure 6. Bid-based model: Separate risk analysis of one objective

For the *SLA* objective in Set A (Figure 6c) and Set B (Figure 6d), FirstReward has the worst performance, but the best volatility as it accepts fewer jobs than the other policies. This is because FirstReward is more risk-averse when considering unbounded penalty and thus accepts fewer jobs to reduce the possibility of incurring penalty. Another possible reason is that FirstReward does not support backfilling and thus may accept fewer jobs compared to FCFS-BF and EDF-BF.

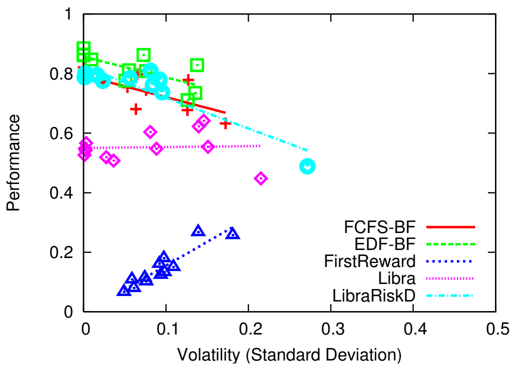
FCFS-BF, EDF-BF, and Libra have constant performance with increasing volatility (zero gradient) in Set B which is worse, compared to having increasing performance with decreasing volatility (decreasing gradient) in Set A. However, LibraRiskD is able to maintain increasing performance with decreasing volatility (decreasing gradient) to record the best performance in Set A and Set B. Although LibraRiskD also has the worst volatility among all the policies, the maximum volatility is only caused by a single point deviation and therefore only applies for one or a few scenarios. It is clear that the main concentration of points for LibraRiskD is close to the best ideal performance and volatility of 1 and 0 respectively. This thus shows that LibraRiskD is able to manage the inaccuracy of runtime estimates a lot better than the other policies.

For the *reliability* objective in Set A (Figure 6e) and Set B (Figure 6f), FCFS-BF and EDF-BF has the best ideal performance and volatility of 1 and 0 respectively due to their generous admission control examining and accepting new jobs only after the previously accepted jobs are completed. FirstReward has the worst performance in Set A and Set B because it delay previously accepted jobs to accommodate new jobs if the new jobs can still return higher utility after taken into consideration the penalties incurred for delaying the previously accepted jobs. But, FirstReward has slightly higher performance and volatility in Set B than in Set A.

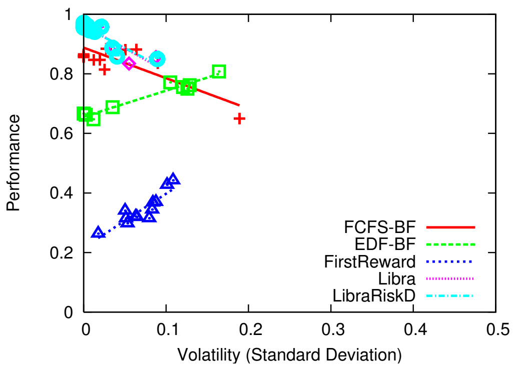
In Set A, Libra and LibraRiskD have a single point deviation due to the scenario of inaccuracy of runtime estimates. In Set B, LibraRiskD achieves higher performance and volatility than Libra, but the higher volatility is only through a single point deviation. Moreover, LibraRiskD has increasing performance with decreasing volatility (decreasing gradient), whereas Libra has constant performance with increasing volatility (zero gradient). Therefore, LibraRiskD is able to handle the inaccuracy of runtime estimates better than Libra.



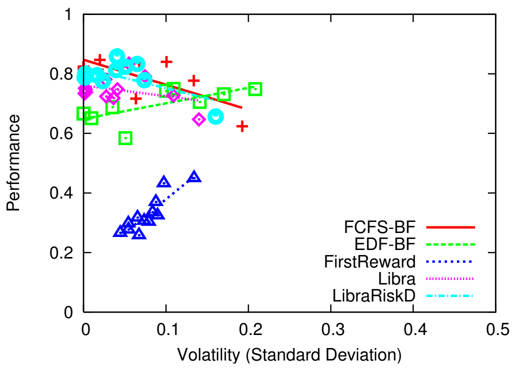
a. Set A: *SLA*, *reliability*, *profitability*



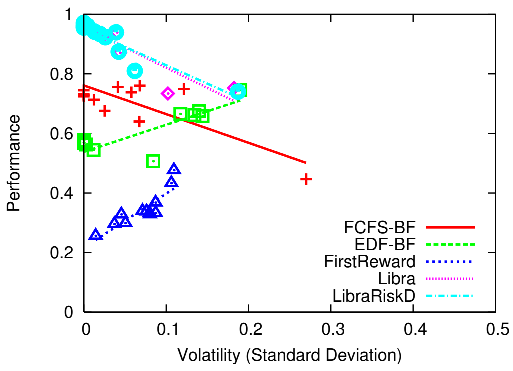
b. Set B: *SLA*, *reliability*, *profitability*



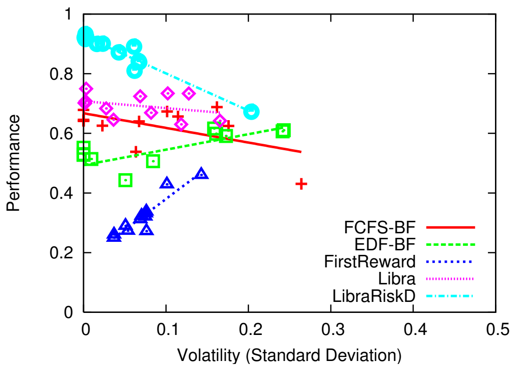
c. Set A: *wait*, *reliability*, *profitability*



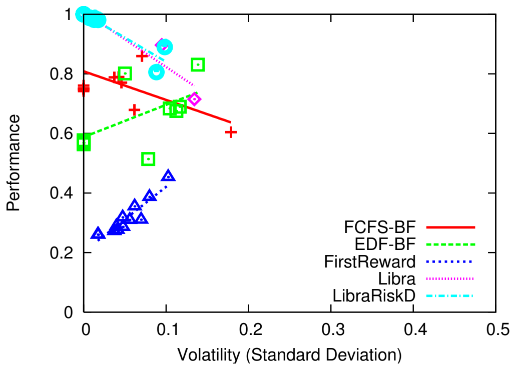
d. Set B: *wait*, *reliability*, *profitability*



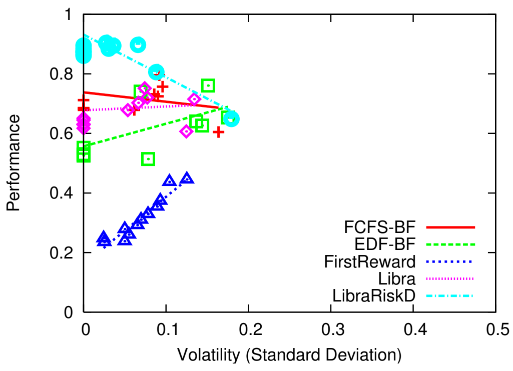
e. Set A: *wait*, *SLA*, *profitability*



f. Set B: *wait*, *SLA*, *profitability*

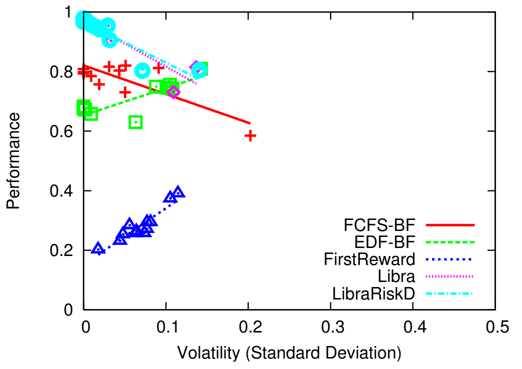


g. Set A: *wait*, *SLA*, *reliability*

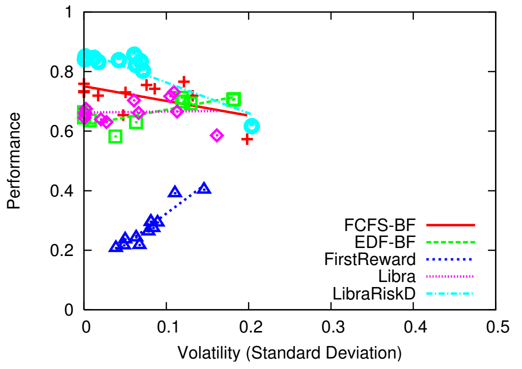


h. Set B: *wait*, *SLA*, *reliability*

Figure 7. Bid-based model: Integrated risk analysis of three objectives



a. Set A: *wait*, *SLA*, *reliability*, *profitability*



b. Set B: *wait*, *SLA*, *reliability*, *profitability*

Figure 8. Bid-based model: Integrated risk analysis of all four objectives

For the *profitability* objective in Set A (Figure 6g) and Set B (Figure 6h), FirstReward has the worst performance, but the best volatility. In addition, FirstReward has increasing performance with increasing volatility (increasing gradient) which is worse than all the other policies which have increasing performance with decreasing volatility (decreasing gradient). This is due to FirstReward being more risk-averse by accepting fewer jobs and not supporting backfilling. All the other policies also have similar volatility. EDF-BF and FCFS-BF have the best performance, followed by Libra and LibraRiskD, but in Set B, LibraRiskD has a marginally higher performance than Libra.

Figure 7 shows the integrated risk analysis of three objectives in Set A (accurate runtime estimates) and Set B (actual runtime estimates from the trace) for the bid-based model. For the four possible combinations of objectives in Set A (Figure 7a, 7c, 7e, and 7g), LibraRiskD has similar performance and volatility as Libra. For the three combinations of objectives that include the *SLA* objective in Set B (Figure 7b, 7f, and 7h), LibraRiskD has considerably better performance, but slightly worse volatility than Libra. This reflects that LibraRiskD is able to perform better than Libra through the *SLA* objective when the runtime estimates are inaccurate.

FirstReward has the worst performance, but the best volatility for the four possible combinations of objectives in Set A and Set B. For the combinations of objectives without *wait* (Figure 7a and 7b) and *SLA* (Figure 7c and 7d) objectives, the difference in performance between FirstReward and the other policies is much greater, as compared to the combinations of objectives without *reliability* (Figure 7e and 7f) and *profitability* (Figure 7g and 7h) objectives. This means that FirstReward performs worse than the other policies due to *wait* and *SLA* objectives.

Figure 8 shows the integrated risk analysis of all four objectives in Set A (accurate runtime estimates) and Set B (actual runtime estimates from the trace) for the bid-based model. In Set A (Figure 8a), Libra and LibraRiskD have the same best performance which is higher than FCFS-BF, EDF-BF, and FirstReward. They also have slightly worse volatility than FirstReward which has the best volatility. But, Libra and LibraRiskD are better than FirstReward since the points of Libra and LibraRiskD are concentrated around the best ideal performance and volatility of 1 and 0 respectively, while the points of FirstReward are evenly spread. In addition, Libra and LibraRiskD have increasing performance with decreasing volatility (decreasing gradient), whereas FirstReward has increasing performance with increasing volatility (increasing gradient).

However, in Set B (Figure 8b), Libra has worse performance than FCFS-BF and EDF-BF, as compared to LibraRiskD which is still able to maintain the best performance. Libra is thus greatly affected by the inaccuracy of runtime estimates, while LibraRiskD is able to handle the inaccuracy of runtime estimates. LibraRiskD also has more points closer to the best ideal performance and volatility of 1 and 0 respectively than the other policies.

7. Conclusion

This paper describes four essential objectives that need to be considered by a commercial computing service in order to realize utility computing: (i) manage wait time for SLA acceptance, (ii) meet SLA requests, (iii) ensure reliability of accepted SLA, and (iv) attain profitability. Two evaluation methods called separate and integrated risk analysis are then proposed to examine whether resource management policies are able to achieve the objectives.

Simulation results have shown that both separate and integrated risk analysis enables the detailed study of various policies with respect to the achievement of a single objective and a combination of objectives respectively. In particular, an objective that is not achieved can severely impact on the overall achievement of other objectives. It is thus essential to examine the achievement of all key objectives together, rather than each standalone objective to correctly identify the best policy that can meet all the objectives. As such, the following summary about the various policies focuses on the achievement of all four objectives together (i.e. integrated risk analysis of all four objectives). Simulation results also reveal that the inaccuracy of actual runtime estimates from the trace can adversely affect the achievement of objectives by non-preemptive policies. In the SDSC SP2 trace used in the simulation, only 8% of the runtime estimates are under estimates, while the remaining 92% are over estimates.

For the commodity market model, Libra+$ is the best policy when runtime estimates are assumed to be accurate. Libra+$ returns significantly higher utility for the commercial computing service even though fewer jobs are accepted by increasing its pricing as workload increases. But, Libra+$ performs worse than FCFS-BF, SJF-BF, and EDF-BF when the actual runtime estimates from the trace is highly inaccurate. For the bid-based model, LibraRiskD is the best policy as it can achieve the best performance even when the runtime estimates from the trace are inaccurate. This is because LibraRiskD not only accepts more jobs given that runtime estimates are often over estimated, but also consider the risk of deadline delay for runtime estimates that are under estimated. Thus, Libra+$ and LibraRiskD are shown to be more effective than Libra for the commodity market model and bid-based model respectively.

FCFS-BF, SJF-BF, and EDF-BF are also less affected by the inaccuracy of runtime estimates as they keep submitted jobs in queues and accept them only prior to execution. Another advantage of this approach is the better selection choice from more jobs in the queue. However, the tradeoff is the longer wait time for SLA acceptance of jobs that lowers the performance of the *wait* objective. There is also minimal impact on FirstReward since it is already more risk-averse when considering unbounded penalty and thus accepts fewer jobs to reduce the possibility of incurring penalty. But, FirstReward has the worst performance if the runtime estimates are accurate.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable comments. This work is partially supported by a Discovery Project grant from the Australian Research Council (ARC) and an International Science Linkage grant from the Australian Department of Education, Science and Training (DEST).

References

1. Altair Grid Technologies: 2000, ‘OpenPBS Release 2.3 Administrator Guide’. http://www.openpbs.org/docs.html.
2. Amazon: 2007, ‘Elastic Compute Cloud (EC2)’. http://www.amazon.com/ec2/.

3. Baker, M. and R. Buyya: 1999, 'Cluster Computing: The Commodity Supercomputer'. *Software: Practice and Experience* **29**(6), 551–576.
4. Buyya, R. and M. Murshed: 2002, '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.
5. Chun, B. N. and D. E. Culler: 2002, '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)*. Berlin, Germany, pp. 22–30.
6. Crouhy, M., D. Galai, and R. Mark: 2006, *The Essentials of Risk Management*. New York, NY: McGraw-Hill.
7. Etsion, Y. and D. Tsafrir: 2005, 'A Short Survey of Commercial Cluster Batch Schedulers'. Technical Report 2005-13, School of Computer Science and Engineering, The Hebrew University of Jerusalem.
8. Feitelson, D. G.: 2007, 'Parallel Workloads Archive'. http://www.cs.huji.ac.il/labs/parallel/workload/.
9. Foster, I. and C. Kesselman (eds.): 2003, *The Grid 2: Blueprint for a New Computing Infrastructure*. San Francisco, CA: Morgan Kaufmann.
10. HP: 2007, 'Utility Computing'. http://www.hp.com/go/utility/.
11. IBM: 2007, 'On Demand Business'. http://www.ibm.com/ondemand/.
12. Irwin, D. E., L. E. Grit, and J. S. Chase: 2004, 'Balancing Risk and Reward in a Market-based Task Service'. In: *Proceedings of the 13th International Symposium on High Performance Distributed Computing (HPDC13)*. Honolulu, HI, pp. 160–169.
13. Islam, M., P. Balaji, P. Sadayappan, and D. K. Panda: 2004, 'Towards Provision of Quality of Service Guarantees in Job Scheduling'. In: *Proceedings of the 6th IEEE International Conference on Cluster Computing (CLUSTER 2004)*. San Diego, CA, pp. 245–254.
14. Kannan, S., M. Roberts, P. Mayes, D. Brelsford, and J. F. Skovira: 2001, *Workload Management with LoadLeveler*. Poughkeepsie, NY: IBM Redbooks. http://www.redbooks.ibm.com/redbooks/pdfs/sg246038.pdf.
15. Kleban, S. D. and S. H. Clearwater: 2004a, 'Computation-at-Risk: Assessing Job Portfolio Management Risk on Clusters'. In: *Proceedings of the 3rd International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO 2004), 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*. Santa Fe, NM.
16. Kleban, S. D. and S. H. Clearwater: 2004b, 'Computation-at-Risk: Employing the Grid for Computational Risk Management'. In: *Proceedings of the 6th IEEE International Conference on Cluster Computing (CLUSTER 2004)*. San Diego, CA, pp. 347–352.
17. Lifka, D. A.: 1995, 'The ANL/IBM SP Scheduling system'. In: *Proceedings of the 1st Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 1995)*, Vol. 949/1995 of *Lecture Notes in Computer Science (LNCS)*. Santa Barbara, CA, pp. 295–303.
18. Moeller, R. R.: 2007, *COSO Enterprise Risk Management: Understanding the New Integrated ERM Framework*. Hoboken, NJ: John Wiley and Sons.
19. Mu'alem, A. W. and D. G. Feitelson: 2001, 'Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling'. *IEEE Trans. Parallel Distrib. Syst.* **12**(6), 529–543.

20. Pfister, G. F.: 1998, *In Search of Clusters.* Upper Saddle River, NJ: Prentice Hall PTR, second edition.
21. Platform Computing: 2001, 'LSF Version 4.1 Administrator's Guide'. http://www.platform.com/services/support/.
22. Popovici, F. I. and J. Wilkes: 2005, 'Profitable services in an uncertain world'. In: *Proceedings of the 18th ACM/IEEE Conference on Supercomputing (SC 2005).* Seattle, WA.
23. Schneider, B. and S. S. White: 2004, *Service Quality: Research Perspectives.* Thousand Oaks, CA: Sage Publications.
24. Sherwani, J., N. Ali, N. Lotia, Z. Hayat, and R. Buyya: 2004, 'Libra: a computational economy-based job scheduling system for clusters'. *Software: Practice and Experience* **34**(6), 573–590.
25. Sulistio, A., G. Poduval, R. Buyya, and C.-K. Tham: 2007, 'On incorporating differentiated levels of network service into GridSim'. *Future Generation Computer Systems* **23**(4), 606–615.
26. Sun Microsystems: 2002, 'Sun ONE Grid Engine, Administration and User's Guide'. http://gridengine.sunsource.net/project/gridengine/documentation.html.
27. Sun Microsystems: 2007, 'Sun Grid'. http://www.sun.com/service/sungrid/.
28. Tsafrir, D., Y. Etsion, and D. G. Feitelson: 2005, 'Modeling User Runtime Estimates'. In: *Proceedings of the 11th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2005)*, Vol. 3834/2005 of *Lecture Notes in Computer Science (LNCS).* Cambridge, MA, pp. 1–35.
29. University of Wisconsin-Madison: 2004, 'Condor Version 6.7.1 Manual'. http://www.cs.wisc.edu/condor/manual/v6.7/.
30. Van Looy, B., P. Gemmel, and R. Van Dierdonck (eds.): 2003, *Services Management: An Integrated Approach.* Harlow, England: Financial Times Prentice Hall, second edition.
31. Xiao, L., Y. Zhu, L. M. Ni, and Z. Xu: 2005, 'GridIS: An Incentive-based Grid Scheduling'. In: *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS 2005).* Denver, CO.
32. Yeo, C. S. and R. Buyya: 2006a, 'A Taxonomy of Market-based Resource Management Systems for Utility-driven Cluster Computing'. *Software: Practice and Experience* **36**(13), 1381–1419.
33. Yeo, C. S. and R. Buyya: 2006b, 'Managing Risk of Inaccurate Runtime Estimates for Deadline Constrained Job Admission Control in Clusters'. In: *Proceedings of the 35th International Conference on Parallel Processing (ICPP 2006).* Columbus, OH, pp. 451–458.
34. Yeo, C. S. and R. Buyya: 2007a, 'Integrated Risk Analysis for a Commercial Computing Service'. In: *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007).* Long Beach, CA.
35. Yeo, C. S. and R. Buyya: 2007b, 'Pricing for Utility-driven Resource Management and Allocation in Clusters'. *International Journal of High Performance Computing Applications* **21**(4), 405–418.
36. Yeo, C. S., R. Buyya, M. D. de Assuncao, J. Yu, A. Sulistio, S. Venugopal, and M. Placek: 2007, 'Utility Computing on Global Grids'. In: H. Bidgoli (ed.): *The Handbook of Computer Networks.* Wiley, Chapt. 143.