

QoS-GRAF: A Framework for QoS based Grid Resource Allocation with Failure provisioning

Gargi Dasgupta, Koustuv Dasgupta, Amit Purohit, and Balaji Viswanathan
IBM, India Research Lab
{gdasgupt,kdasgupta,ampurohi,bviswana}@in.ibm.com

Abstract

The emergence of utility computing has led to a growing interest in extending the usability of grid technologies beyond scientific computing, towards a more service centric model. From the customer's viewpoint, this model promises on-demand delivery of IT capabilities and usage based pricing schemes. Grid providers on the other hand attempt to capitalize on this trend, by exploring innovative ways of maximizing their revenues while ensuring high utilization of resources. Maximization of business revenue requires differentiation of services based on QoS criteria. Existing approaches on resource provisioning in the grid are designed to handle static customer priorities with simplistic pricing policies. In reality, however, grid SLAs can express multiple resource requirements with different QoS pricing and complex co-selection dependencies. In addition, resource allocation in the grid does not address the problem of meeting service guarantees, in case of resource downtime. Since many of these SLAs have very high breach penalty clauses, service guarantees need to be satisfied even in case of failures. To this end, we present QoS-GRAF, a framework for QoS based Grid Resource Allocation with Failure provisioning. The proposed framework exploits the notion of revenue maximization based on service differentiation, along with advanced reservation to handle failures. We design intelligent algorithms to solve the grid resource allocation problem and analyze their performance in terms of earned revenue, resource utilization and revenue loss resulting from failures.

1 Introduction

For a long time grid computing was viewed as the key technology for harnessing the compute power of high performance virtual supercomputers connected across a WAN. However, with the emergence of the utility grid computing paradigm, it has become increasingly desirable for organizations worldwide to improve efficiency by outsourcing their business processes, and for providers to supply grid capacity with the primary goals of charging for service usage and high utilization of shareable computing. IBM's Deep Computing Center on Demand (DCCoD) and the Sun Grid are examples of on-demand grids providing large and small companies access to their supercomputing power.

In this utility model, the success of the customer-provider relation is largely dependent on whether the business partner can be trusted to provide an on-time reliable service. To ensure this quality of service, the provider and the customer mutually decide on a service level agreement (SLA) that specifies among other things, base expectations from the service provider, differentiated pricing policies based on different levels of QoS, and breach clauses specifying penalties to be paid in case base requirements are not met. A successful and competitive provider must be able to maintain customer goodwill by satisfying the demand for high-quality service, and at the same time constantly look towards improving its efficiency via business process management technologies. At the heart of these technologies lies the problem of business revenue maximization. Differentiated pricing schemes offered by customers, associated with different levels of QoS expectations, empowers the provider with the capability of business revenue maximization by way of quality of service differentiation.

SLA-based business profit maximization has been studied for web servers[20]. However, in the grid environment there are additional constraints that need to be addressed. A grid job is typically dependent on multiple resources for its completion (e.g. CPU, disk, bandwidth) and each of these *dependencies* must be satisfied for the job to execute. [18] proposes a co-selection solution, where business values of jobs are interpreted from their priorities. However, in general, a job could have different QoS requirements (with different price offerings) for each of its dependencies. Therefore, assigning static priorities to jobs over-simplifies the problem. [3] proposes a business process management framework that enables optimizing the business performance of executing workflows by associating a business-value function with each process. Once again, resource co-selection requirements are not addressed. In reality, customer SLAs may reflect varying degrees of importance with different resource dependencies. [15] proposed an SLA-based approach to scheduling workflows for grid environments. Tangible research results of the effort are not available, and the proposal lacks the notion of optimizing business goals. In addition, none of these techniques address the issue of provisioning for failures. In order to avoid paying heavy breach penalties, QoS guarantees need to be met by providers even in case of resource failures.

We propose a framework for QoS based Grid Resource Allocation with Failure provisioning (QoS-GRAF), that advocates optimizing business metrics based on SLA driven pricing policies. Our work is most relevant to service oriented grid computing and failure provisioning. Within the framework, we focus on designing revenue maximization algorithms – where given a batch of jobs, their SLA specifications for multiple dependencies, and the set of available resources in the grid, our objective is to (a) assign a *primary* resource for each job such that total revenue earned is maximized, and (b) assign a *backup* resource for each job such that the total revenue loss is minimized in the case of a failure. To the best of our knowledge, this is the first approach that marries the idea of SLA based business performance optimization in utility computing grids with the notion of failure provisioning.

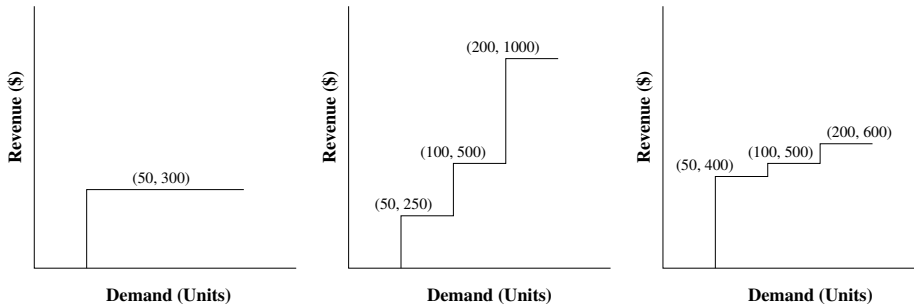


Figure 1: Example demand-revenue step functions

2 Business Profit Maximization in the Utility Grid

Before we formulate the problem, it is important to outline the key design issues involved in providing business profit maximization for grid service providers.

2.1 SLA-based Service Differentiation

SLAs are emerging as the standard concept based on which the grid utility computing model is shaping up [8]. An SLA captures the high-level service requirements, commonly termed as Service Level Objectives (SLOs), as well as the business value associated with satisfying these requirements. An example SLO of response time and its business values could be (response time < 10ms, revenue earned = 1000\$), (response time between 10-15ms, revenue earned = 300\$) and (response time > 15 ms, revenue earned = 0\$). In essence, the service level objectives along with pricing serve as an incentive for providers to perform service differentiation and revenue maximization.

2.1.1 Providing QoS Guarantees

In order to earn business revenue, providers need to satisfy customer SLOs with certain guarantees. However, in order to provide such guarantees, the SLOs need to be translated into corresponding resource requirements, that can be monitored, managed, and provisioned. There exists a body of research work [11, 14] that identifies the gap between business level metrics and resource level requirements. Commercial tools have also been developed [4, 9] to translate the business metrics into lower level resource requirements. These requirements can then be satisfied using provisioning techniques. Several solutions have been proposed [6, 12] that deliver end-to-end QoS guarantees using *advance reservation*, and work well for low-level resource requirements like MIPS, network bandwidth, disk I/O operations etc. In the same context, solutions have been proposed [1, 2] for supporting application-specific reservations on shared computing resources.

2.1.2 Demand-Revenue Model

Given the resource level requirements and the business values associated with satisfying the requirements, it is possible to determine a *demand-revenue* function that captures the relationship between the demand for a certain level of QoS and the price paid to a provider for meeting it. In

Resource	Revenue
Base Storage	1000\$ for 200 MB
Incremental Storage	100\$/MB upto 1Gb
Base Bandwidth	1500\$ for 10 Mbps
Incremental Bandwidth	25\$/Mbps upto 1Gbps

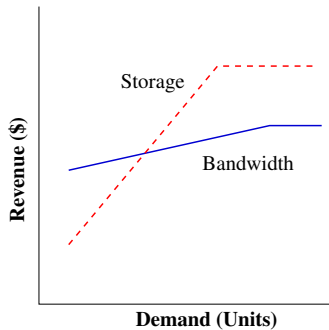


Figure 2: Resource level SLOs of an application with network and storage dependencies

general, the demand-revenue function can be represented as $\mathbb{F} = f(d)$, where \mathbb{F} and d represent the revenue and the demand, respectively. We consider demand-revenue functions that have non-decreasing revenue with increase in demand. From the customer's point of view, it is intuitive to adopt a step-wise demand-revenue function [20]. Therefore, the demand-revenue step function can be written as

$$\mathbb{F}(d) = \alpha_1 f_1(d) + \alpha_2 f_2(d) + \dots \alpha_n f_n(d), \quad (1)$$

where

$$\alpha_i \in \mathbb{R}, f_i(d) = 1 \text{ if } d \in [a_i, b_i), 0 \text{ otherwise, for } i = 1, \dots, n \quad (2)$$

We assume that a job has a demand-revenue function associated with each resource dependency. Each step in this function represents a particular QoS level. The revenue at each QoS level i represents how much a customer is willing to pay in order for its demand at that level to be satisfied. The demand-revenue function captures the essence of pricing based service differentiation. Figure-1 illustrates some sample demand-revenue step functions.

2.1.3 Multiple Dependencies

Given a heterogeneous pool of resources in a grid (e.g. CPU, storage, network bandwidth, database etc.), a dependency of a job indicates its requirement for a particular resource type. A job could have multiple dependencies. Typically, some of these dependencies might be more important to the customer than others, and hence s/he might be willing to pay more per unit price for these. Thus, it is reasonable to assume that different dependencies of an application may have different associated demand-revenue functions. Figure-2 shows resource level SLOs of an application and the corresponding demand-revenue functions for the two dependencies of storage and network sub-system. As is evident from the demand-revenue function, this customer has a higher base price for network bandwidth but is willing to pay a higher per unit storage price.

2.2 Failure Provisioning

QoS service guarantees need to be met even in case of resource failures/downtime, which are not uncommon in the grid composed of heterogeneous resources and services. Since SLAs often have high penalty clauses, it is imperative for grid service providers to additionally provision resources

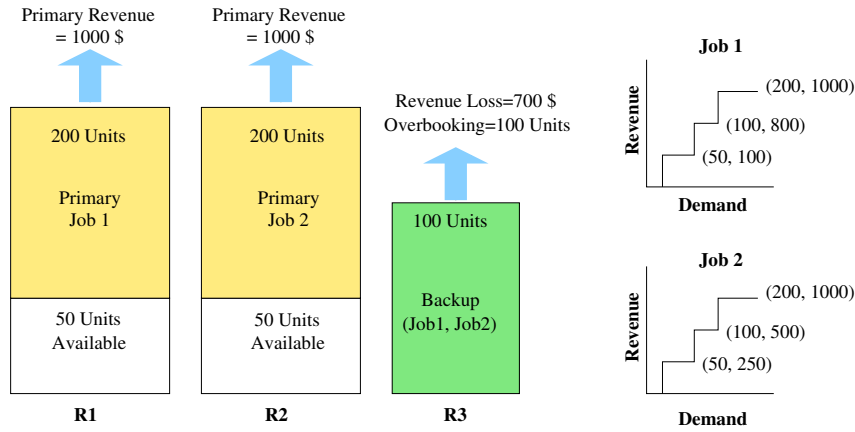


Figure 3: Backup Sharing

for failures. Failure provisioning in grids can be either a re-active or a pro-active strategy. In the re-active strategy, the decision of which resources will handle the failed job requests is taken after the failure strikes. However, the discovery and provisioning time of a potential resource to host the failed job, or the unavailability of such a resource at failure time, may result in jobs missing their base service goals causing high revenue losses. In the pro-active scheme, it is decided apriori where a job will be migrated to in case there is a failure, and resources are provisioned for it. So in case of a failure, a switch to the backup resource can be guaranteed and made efficient. In the context of business revenue maximization, pro-active failure provisioning reduces revenue loss of a provider. QoS pricing policy based provisioning helps high revenue earning jobs to be backed up at a higher QoS levels than their lower revenue counterparts.

The central issue in pro-active failure provisioning is that how much resources do you allocate as backup for a job, assuming that the job is currently running with some primary resources. Since backup resources need to be *disjoint* from the corresponding primary allocations, each job will now have a primary and a backup allocation, for each dependency. The amount of resources dedicated for backup reservation represents the percentage of overbooking in the system. In the most conservative approach, one would reserve as backup the minimum resources required by the job to execute in the system. However, this may result in large revenue loss. On the other extreme, no revenue loss will be incurred if the entire amount of primary resources is overbooked. But, given limited backup resources, it might be practically infeasible to design a system having 100% overbooking. Thus, it is important to find the backup allocation that minimizes revenue loss without excess overbooking.

2.2.1 Backup Sharing

Overbooking in a system can be reduced by sharing backup resources among jobs that can be guaranteed never to fail simultaneously. Consider a system with three resource R1, R2 and R3 with capacities 250, 250 and 100 units respectively, and two jobs with demand-revenue functions

as shown in Figure-3. Job 1 runs on R1 with a primary reservation of 200 units, earning 1000\$ and job 2 runs on R2 at its maximum requirement (200 units for 1000\$). Assume the single failure model, where either R1 or R2 (but not both) can fail. In the absence of other information, both failures have equal likelihood and we need to protect both jobs against any one of them. The approach we take is to provision backups such that the total revenue loss is minimized. When no backup sharing is allowed, the best option is to backup job 1 at level 2 on R3 and job 2 at level 1 on R1. This may incur a worst case revenue loss of 750\$ and an overbooking of 150 units. But with backup sharing, both jobs can share backup capacities on R3, i.e. both jobs can be backed up at level 2 on R3 incurring a worst case revenue loss of 500\$ and overbooking of 100 units.

2.3 Tying the knot: Service Differentiation with Failure Provisioning

SLA based service differentiation and failure provisioning are two important principles we consider for business profit maximization in the grid computing model. Demand-revenue functions capture differentiated pricing schemes for desired level of QoS. To maximize business profits, providers should offer service differentiation by way of advanced resource provisioning based on these QoS pricing schemes. Additionally, to reduce revenue loss in case of failures, failure provisioning should be done based on these pricing schemes. Given a set of customer jobs, their corresponding SLAs, the demand-revenue functions for all dependencies, and the total available resources, we focus on the problem of primary and backup allocation, such that the business revenue is maximized.

3 QoS-GRAF Framework

In this section, we present our framework, QoS based Grid Resource allocation with failure provisioning (QoS-GRAF). Figure-4 gives a detailed view of the proposed framework. We consider as input a set of grid jobs with their SLA descriptions. An **SLA Manager** maintains these SLAs, and maps their application level requirements to the corresponding resource level requirements. It also updates the **Utility Database** with the demand-revenue functions of all jobs for all dependencies. The **QoS-GRAF Controller** manages execution of all unfinished jobs such that their base requirements are met. In addition, it enforces the scheduling activities and the resource allocation decisions. The jobs are admitted to the system via an **Admission Controller** that consults business policies, customer reputation, current state of system etc. to admit/reject jobs. Once a job is admitted, it is added to the list of jobs in the **Batch Scheduler** queue. From job start/end times and other meta-information, the batch scheduler computes a temporal schedule of jobs and forms batches of jobs of similar duration of activity that will be competing for the same set of resources. For each batch of jobs and their resource requirements, the QoS-GRAF controller invokes the primary and backup allocators to compute the set of allocations that will be reserved in advance. The **Resource Provisioning Manager (RPM)** actually reserves the above resources and maintains bookkeeping information about a job's primary and backup allocations. Monitoring

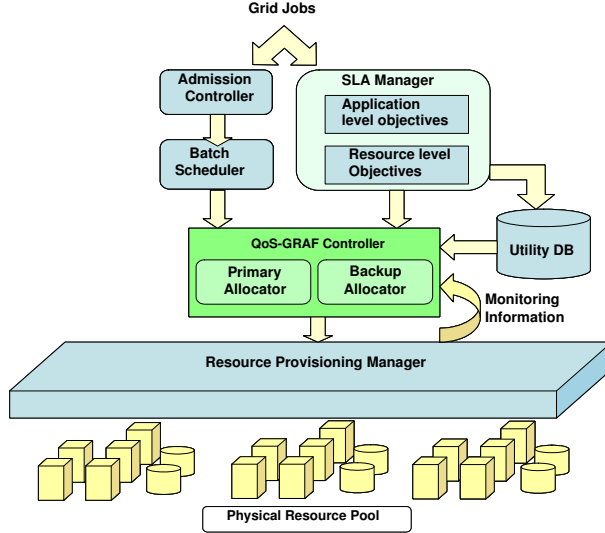


Figure 4: QoS-GRAF Framework

information is fed back to the QoS-GRAF controller. Backfilling techniques [13] are used by the controller to increase system utilization. Once a failure happens, the failure is escalated to the QoS-GRAF controller by the monitoring feedback. The controller then triggers the RPM to switch the set of concerned jobs to their backups. One inherent assumption in this framework is that it is possible to transfer the state of a failed job to its backup resource in a timely and efficient manner. In some cases, this might require process migration and checkpointing techniques [10]. In case, it is not possible to resume the failed job from its checkpointed state, it can simply be restarted at the backup. Note that, the cost of job migration is not considered during backup allocation and is beyond the scope of this work.

4 Problem Formulation

We next address the central problem of provisioning primary and backup resources for a batch of jobs. We consider a grid with J heterogeneous resources. The input to the system is a batch of K jobs, where each job has *atmost* D resource dependencies (e.g. CPU, disk, DB). Each resource can satisfy exactly one dependency. The demand-revenue function of a job, for each dependency, is a step-wise function with I distinct steps. Each step represents a different QoS level that can be used to service the job. Further, we assume that the available resource capacity is sufficient to guarantee the minimum requirement for each dependency of a job.

We use the the following notations: (1) $D_\delta(i, k)$ denotes demand of job k at level i for dependency δ , (2) $R_\delta(i, k)$ denotes the revenue earned by job k if demand $D_\delta(i, k)$ is satisfied, (3) $Dmin_\delta(k)$ and $Dmax_\delta(k)$ denote the minimum and maximum demand of job k for dependency δ respectively, (4) $C^P(j)$ and $C^B(j)$ denote available capacity of resource j for primary and backup allocations

respectively, (5) $Sat_\delta(j)$ is 1 if resource j satisfies dependency δ , and 0 otherwise. Given a batch of jobs with multiple dependencies and the demand-revenue function associated with them, we first formulate the revenue maximization problem for the primary allocation.

4.1 Maximum Revenue Primary Allocation (MRPA)

Let $U_\delta^{i,j,k}$ be an indicator variable, assigned 1 if job k is allocated primary resource j , at level i , for dependency δ ; 0 otherwise.

The Maximum Revenue Primary Allocation problem can be then stated as follows:

Objective function

$$\max \sum_{\delta=1}^D \sum_{i=1}^I \sum_{j=1}^J \sum_{k=1}^K U_\delta^{i,j,k} Sat_\delta(j) R_\delta(i, k), \quad (3)$$

subject to

Capacity Constraint:

$$\sum_{i=1}^I \sum_{k=1}^K U_\delta^{i,j,k} Sat_\delta(j) D_\delta(i, k) \leq C^P(j), \quad \forall j \quad (4)$$

Minimum Requirement Constraint:

$$\sum_{i=1}^I \sum_{j=1}^J U_\delta^{i,j,k} Sat_\delta(j) D_\delta(i, k) \geq Dmin_\delta(k), \quad \forall \delta, k \quad (5)$$

Maximum Requirement Constraint:

$$\sum_{i=1}^I \sum_{j=1}^J U_\delta^{i,j,k} Sat_\delta(j) D_\delta(i, k) \leq Dmax_\delta(k), \quad \forall \delta, k \quad (6)$$

Feasible Assignment Constraint:

$$\sum_{i=1}^I \sum_{j=1}^J U_\delta^{i,j,k} Sat_\delta(j) \leq 1 \quad \forall \delta, k \quad (7)$$

Integrality Constraint:

$$U_\delta^{i,j,k} \in \{0, 1\}, \quad \forall \delta, i, j, k. \quad (8)$$

The capacity constraint ensures that resource capacities are not exceeded. The minimum and maximum requirement constraint ensures that the minimum demand is met for each dependency and that the allocation does not exceed the maximum demand of any dependency. The feasibility assignment constraint makes sure that each job is assigned to exactly one resource and at exactly one priority level. Finally, the integrality constraint prevents jobs from being split across resources and priority levels. This is a mixed integer programming problem, which can be shown to be NP-hard by reduction from the multi-bin knapsack problem [7]. We describe a heuristic approach to solve the problem in a later section.

Next, we consider the problem of backup allocation for each job, i.e., finding a backup resource for each primary. Note that, the integrated version of the problem where all primary and backup allocations are computed simultaneously is hard to solve. For ease of handling, we consider the

decoupled problem, where given a set of primary allocations (one for each job), we need to compute a backup allocation for each primary, such that the total revenue loss is minimized across all jobs.

Before we detail the problem formulation, we wish to point out that our approach assumes that no more than one failure occurs at a given point of time. Further, the mean-time-between-failures is usually higher than the mean-recovery-time. This, in turn, implies that we need to find backup resources that can continue to provide some QoS to all the jobs during this recovery time. We also note that providing backup protection against k multiple failures is a much harder problem. In the case of advanced reservation, k backup resources need to be allocated for each primary. Given that the probability of k failures decreases rapidly as k increases, the single failure model has received the maximum attention in the research community [17]. We assume a single failure model in our formulation, and later provide insights for handling multiple failures.

4.2 Minimum Loss Backup Allocation (MLBA)

Given a set of jobs with multiple dependencies, the demand-revenue functions, and the primary allocation for each job—the problem is to find a set of backup allocations, such that the total revenue loss, in moving each primary to a backup allocation is minimized. In a single failure model, all primaries are equally likely to fail, and hence minimizing the total revenue loss essentially minimizes the average revenue loss due to a single failure.

We introduce the following notations: (1) RP_δ^k is the total revenue accrued by the primary allocation of job k for dependency δ , (2) DP_δ^k is the primary allocation of job k for dependency δ , (3) $B_\delta^{p,i,j,k}$ is an indicator variable, assigned to 1 if job k , running on primary resource p , is allocated backup resource j at level i , for dependency δ ; and 0 otherwise, (4) r_δ^k is the backup revenue of job k for dependency δ , and (5) α_δ^k be the revenue loss by job k for dependency δ .

We define,

$$r_\delta^k = \sum_{p=1}^J \sum_{j=1}^J \sum_{i=1}^I B_\delta^{p,i,j,k} Sat_\delta(j) R_\delta(i,k), \quad \forall \delta, k \quad (9)$$

$$\alpha_\delta^k = RP_\delta^k - r_\delta^k, \quad \forall \delta, k \quad (10)$$

The Minimum Loss Backup Allocation problem can be then stated as follows:

Objective function

$$\min \sum_{k=1}^K \sum_{\delta=1}^D \alpha_\delta^k, \quad (11)$$

subject to

Minimum Requirement Constraint:

$$\sum_{i=1}^I \sum_{j=1}^J \sum_{p=1}^J B_\delta^{p,i,j,k} Sat_\delta(j) D_\delta(i,k) \geq Dmin_\delta(k), \quad \forall \delta, k \quad (12)$$

Maximum Requirement Constraint:

$$\sum_{i=1}^I \sum_{j=1}^J \sum_{p=1}^J B_\delta^{p,i,j,k} Sat_\delta(j) D_\delta(i,k) \leq DP_\delta^k, \quad \forall \delta, k \quad (13)$$

Feasible Assignment Constraint:

$$\sum_{i=1}^I \sum_{j=1}^J \sum_{p=1}^J B_{\delta}^{p,i,j,k} Sat_{\delta}(j) \leq 1, \quad \forall \delta, k \quad (14)$$

Disjoint Primary Constraint

$$B_{\delta}^{p,i,j,k} = 0, \quad \forall p = j \quad (15)$$

Integrality Constraint

$$B_{\delta}^{p,i,j,k} \in \{0, 1\}, \quad \forall \delta, i, j, k \quad (16)$$

Backup Sharing Constraint

$$S_{jp} \geq \sum_{i=1}^I \sum_{k=1}^K B_{\delta}^{p,i,j,k} Sat_{\delta}(j) D_{\delta}(i, k), \quad \forall \delta, jp \quad (17)$$

Capacity Constraint

$$S_{jp} \leq C^B(j), \quad \text{for all } j \quad (18)$$

The integrality, feasible assignment, and minimum requirement constraints are the same as before. The maximum requirement constrained is now enforced by the primary allocation of the job, i.e., we do not allow a backup to exceed the corresponding primary allocation. The disjoint primary constraint ensures that primary and backup resources of a job are disjoint. The backup sharing constraint ensures that backups of disjoint primaries can share the backup capacity. Finally, the capacity constraint ensures that the total shared backup capacities are within the available limits. Once again, the MLBA problem is a mixed integer programming problem which can be shown to be NP-hard [7].

LPRound-Primary()

1. Sort primary indicators in descending order of real values
2. For each job k, select (i, j, k) with maximum indicator value
 - 2.1. Assign job k to resource j, s.t. $D(i, k) \leq \text{AvailCap}(j)$; Reduce $\text{AvailCap}(j)$
3. While (UPDATE POSSIBLE) UpdatePrimaryAllocation()

UpdatePrimaryAllocation()

1. Sort jobs in descending order of unit revenue increment
2. For job i, if $(\text{AvailCap}(j) \geq D(i+1, k) - D(i, k))$:
 - 2.1. Assign(i+1, j, k), Reduce $\text{AvailCap}(j)$
3. else, find resource j' s.t. $\text{AvailCap}(j') \geq D(i+1, k)$
 - 3.1 Assign(i+1, j' , k), Reduce $\text{AvailCap}(j')$, Increase $\text{AvailCap}(j)$

Figure 5: Heuristic MRPA for primary assignments

LPRound-Backup()

1. Sort backup indicators in descending order of real values
2. For each job k , select (p, i, j, k) with maximum indicator value
 - 2.1. Assign backup of job k to resource j , s.t. $D(i, k) \leq \text{AvailBkupCap}(p, j)$;
 - 2.2. Reduce $\text{AvailBkupCap}(p, j)$
3. While (UPDATE POSSIBLE) $\text{UpdateBackupAllocation}()$

UpdateBackupAllocation()

1. Sort jobs in descending order of unit revenue loss
2. For job i , if $(\text{AvailBkupCap}(p, j) \geq D(i+1, k) - D(i, k))$:
 - 2.1. $\text{AssignBkup}(i+1, j, k)$, Reduce $\text{AvailBkupCap}(j)$
3. else, find resource j' s.t. $\text{AvailBkupCap}(p, j') \geq D(i+1, k)$
 - 3.1 $\text{AssignBkup}(i+1, j', k)$, Reduce $\text{AvailBkupCap}(p, j')$, Increase $\text{AvailBkupCap}(p, j)$

Figure 6: Heuristic MLBA for backup assignments

4.3 Linear Relaxation Based heuristics

In order to develop efficient heuristics, we consider linear relaxations of the above problems. In effect, we relax the integrality constraint so that it can take real values. We then solve the LP problems to obtain the fractional optimal solutions. However, this solution may not be a feasible solution, since it may split jobs across QoS levels and/or resources. To obtain a feasible solution we use a sort-and-round technique based on the real values of the indicators i.e., $U_d(i, j, k)$ and $B_d(p, i, j, k)$. For jobs split across levels, the maximum value of the split indicator is considered. If a job can be fitted on the “indicated” resource at the “indicated” level, then assignment is done. If not, the job is initially allocated to a lower QoS level and marked for later handling. Once the initial assignment for all jobs is complete, an iterative routine tries to update allocations by boosting the QoS of all marked jobs. This is done by either boosting its level at the already assigned resource (in case capacity is available), or assigning it to a different resource, i.e., one that can fit this job at a QoS level higher than its current level. The primary heuristic for Maximum Revenue Primary Allocation (MRPA) is shown in Figure-5.

The backup heuristic employs a similar techniques, but additionally utilizes sharing information. In particular, the available backup capacity, $\text{AvailBkupCap}(p, j)$, indicates the backup capacity available to job k for sharing on resource j , where p is the primary resource of k . The available backup capacity is given $\text{AvailBkupCap}(p, j) = \text{AvailCap}(j) - \text{BkupCap}(p, j)$, where $\text{BkupCap}(p, j)$ is the total backup capacity allocated on resource j by all jobs that have a primary on p . The backup heuristic for Minimum Loss Backup Allocation (MLBA) is shown in Figure-6.

Jobs		Demand (Units)				Revenue (\$)			
		Level 1	Level 2	Level 3	Level 4	Level 1	Level 2	Level 3	Level 4
Small	LO	20	100	200	400	1000	2000	3000	4000
	HI	100	200	400	500	2000	3000	4000	5000
Medium	LO	200	400	500	2000	2000	4000	8000	12000
	HI	400	500	2000	5000	4000	6000	10000	15000
Large	LO	1000	2000	5000	8000	12000	20000	25000	27000
	HI	2000	5000	8000	10000	15000	24000	27000	30000

Table 1: SLA Model

5 Performance Evaluation

In this section, we present experimental results to illustrate the effectiveness of our approach for (i) maximizing earned revenue, and (ii) minimizing revenue loss due to failure. A large number of experiments were conducted under a wide range of parameter settings in a simulated grid environment. Our objective was to assess the performance of the proposed algorithms, MRPA (for primary allocation) and MLBA (for backup allocation), both with respect to the optimal solution and alternative heuristics. A representative set of results are presented herein. We start by describing the alternative heuristics for primary and backup computation, followed by the simulation setup, and finally report our main findings.

5.1 Alternative Heuristics

We compare our approach with four different heuristics. Each of the heuristics, follow a common set of steps: sort the available jobs (in descending order of priority) based on some criterion; starting with the job at the head of the sorted list, select a suitable resource instance that can accommodate the maximum requirement of the job; assign the job to the resource instance by allocating required demand and continue until the job list is empty or there are no resources available. If the job list is empty, the algorithm terminates. Otherwise, in order to make space for minimum requirements of the unassigned jobs, the allocations of previously assigned jobs are reduced. The sorted list is traversed in the reverse order, where jobs higher up in the list are selected only after the allocations of all lower priority jobs have been reduced to their minimum capacity. The algorithm ends when all jobs have resources assigned to them. This procedure is repeated for each dependency. The four heuristics differ in their *sort criterion* and *suitable resource selection method*. In case of backup allocation, the heuristics additionally allow sharing of backup resources.

- **MAX_BIG**: This heuristic sorts the job list based on maximum earned revenue (maximum revenue loss in case of backup) of each job. Among multiple resources that satisfy the job requirement, one with the largest available capacity is chosen.

- **MAX_BEST**: This heuristic is similar to **MAX_BIG** in terms of sorting the job list. Among multiple resources that satisfy the job requirement, one that has its available capacity closest to the demand is chosen.
- **UNIT_BIG**: In this heuristic the sorting order is determined by revenue earned per unit demand (revenue loss per unit demand in case of backup) of the jobs. The choice among multiple resources is made in the same way as **MAX_BIG**.
- **UNIT_BEST**: This heuristic is a combination of **UNIT_BIG** and **MAX_BEST**. It sorts the job list as **UNIT_BIG** does and for choosing the most appropriate resource, it follows the same principle as **MAX_BEST**.

5.2 Simulation Setup

The input to the algorithms is a batch of K jobs, and the set of resources J that the jobs will be competing for. Each job has three resource dependencies—namely, compute server, network sub-system and storage sub-system. Each dependency is associated with a demand-revenue function that specifies multiple QoS levels. We consider four levels of QoS and compare the algorithms in terms of revenue earned (\$) for primary allocations, and revenue loss suffered (\$) for backup allocations. The total available capacity in the system is the sum of available primary and backup capacities, where the backup capacity is some fraction of the primary. Let $\hat{p}f, \hat{b}f$ be the primary and backup resource availability factors, C^P, C^B be the available primary and backup capacities, and \bar{D} be the average resource requirement of a job.

We define the total available capacity in the system as:

$$TotalAvlCap = C^P + C^B = C^P + \hat{b}f C^P \quad (19)$$

where,

$$C^P = \hat{p}f \sum_{jobs} \bar{D}_{job}, \quad \hat{p}f, \hat{b}f \in (0, 1) \quad (20)$$

This gives us the capability of controlling the load imposed on the system by tuning the $\hat{p}f, \hat{b}f$ parameters. The primary capacity of each resource instance is selected from a normal distribution with mean $\mu = TotalAvlCap/J$ and variance $\sigma = 0.75$. The backup capacity depends on the primary capacity and $\hat{b}f$. The set of jobs is divided into three pools of small, medium and large jobs depending on their demand requirements and the corresponding value they are willing to pay at each QoS level. The demand-revenue ranges $[LO, HI)$ at each QoS level are shown in Table-1 for one dependency. The overlapping regions between small, medium and large jobs simulate the fact that there might be some small jobs that are willing to pay more for a higher level of QoS than a medium job at a lower service level. For each job, the demand-revenue at each QoS level is picked randomly with a uniform distribution in the $[LO, HI)$ range. Finally, the mix of small, medium and large jobs is varied for different experiments.

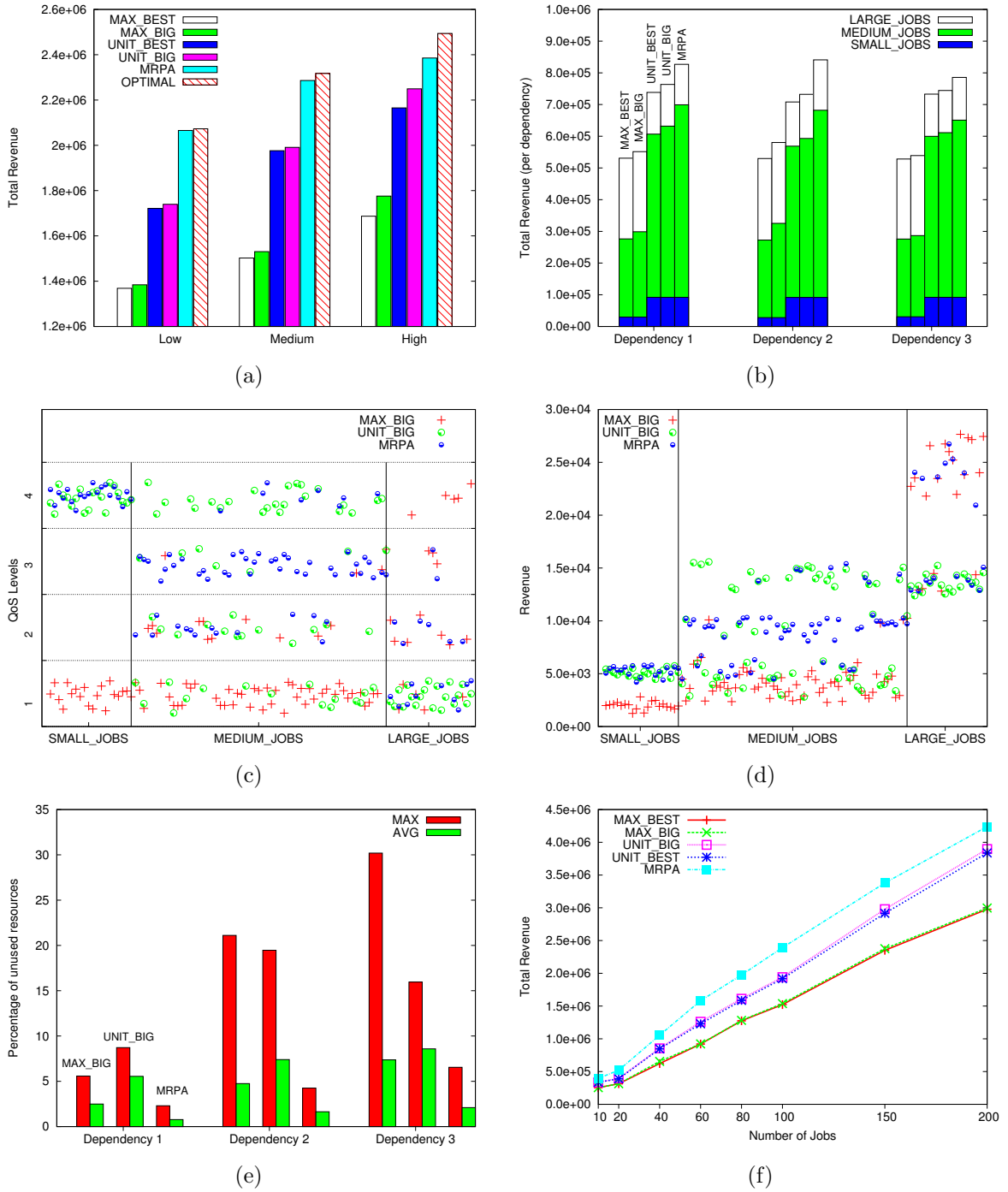


Figure 7: Performance of different heuristics for primary allocation. (a) Total revenue at different \hat{p} (b) Revenue per dependency for small, medium, large jobs (c) Assigned QoS levels for all jobs (d) Revenue earned by all jobs (e) % of unused resources per dependency (f) Total revenue with increasing number of jobs

5.3 Primary Allocation

In the first set of experiments, we evaluate the performance of our MRPA algorithm. For all the runs in this set, the mix of small, medium and large jobs in the system is 20%,70%,10%.

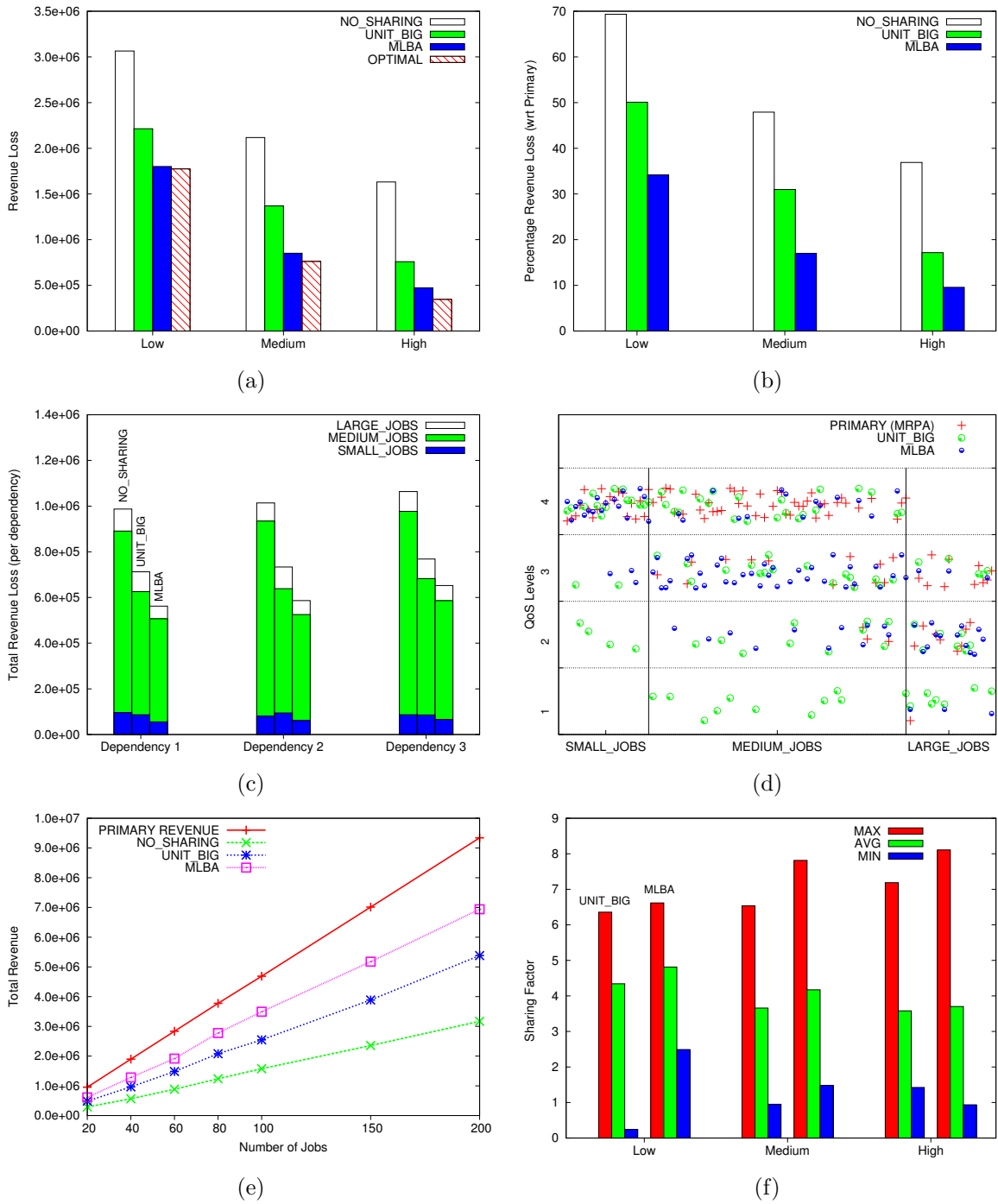


Figure 8: Performance of different heuristics for backup allocation. (a) Total revenue loss at different $\hat{b}f$ (b) % Revenue loss wrt primary revenue (c) Revenue loss per dependency for small, medium, large jobs (d) Assigned QoS levels for all jobs (e) Total backup revenue with increasing number of jobs (f) Sharing of backup resources at different $\hat{b}f$

We define the low, medium and high resource availability conditions, by choosing values of $\hat{p}f$ in the ranges of (0.25-0.5), (0.5-0.75) and (0.75-1.0), respectively. In Figure-7a, with a batch of

100 jobs in the system, we compare the total revenue earned by each algorithm with the optimal (fractional) solution, at various resource availability conditions. We observe that all algorithms earn more revenue at medium and high availability conditions than in lower availability condition, where there is high contention for resources. The UNIT algorithms do consistently better than their MAX counterpart. This reconfirms our claim that the greedy technique of choosing the maximum revenue earning job is not the best solution for profit maximization scenarios. Also, the BIG fit algorithms result in higher revenue earnings than the BEST fit ones. MRPA outperforms UNIT_BIG by about 20% and always achieves performance within 5% of the optimal at the low, medium and high availability conditions.

In Figure-7b, we focus on 100 jobs with $\hat{p}f = 0.75$. The intention is to see the breakup of the revenue earned by small, medium and large jobs across all dependencies. We note that for all algorithms, the major portion of the revenue comes from medium jobs. Both UNIT_BIG and MRPA have an equal contribution in revenue from the small jobs. However, the big difference in the total revenue comes from the medium and large jobs. For deeper understanding, we present the scattered plots in Figure-7c,d showing the QoS levels for the small, medium and large jobs and the revenue earned, for one dependency. The results show that MAX_BIG greedily runs the maximum revenue jobs at the highest level, and hence all the small and medium jobs are pushed to the lowest level. UNIT_BIG earns most of its revenue by running more medium jobs at the highest priority. However, because of this, it is forced to push all large jobs at the lowest priority. In comparison, MRPA runs the medium and large jobs well distributed between QoS levels 2 and 3, with its maximum revenue coming from medium jobs at level 3. Because MRPA is able to adjust priorities of all classes of jobs, it does not penalize any particular class while earning steady revenue across all classes. The total revenue earned by MRPA exceeds that of UNIT_BIG and MAX_BIG by 20% and 30% respectively.

While allocating jobs to resources, it might be that the unused capacity left in a resource is too small to fit in any job requirement. This fragmentation results in lower utilization of available resources. In Figure-7e, the maximum and average unused resources is shown for each heuristic. Figure shows that MRPA achieves more efficient resource utilization (due to lesser fragmentation) than UNIT_BIG and MAX_BIG. On an average with MRPA, about less than 3% resources were unutilized. The maximum capacity unused on any resource instance was no more than 7%. While UNIT_BIG and MAX_BIG achieve an average resource fragmentation of within 10%, their maximum unused capacity is as high as 30%.

Finally, Figure-7f shows how the total revenue earnings scale with the number of jobs, when the total available resources is constant. For a small number of jobs, all algorithms do well. But as the number of jobs in the system increases, MRPA achieves substantial more in revenue than the best among the other heuristics. At about 200 jobs in the system, the revenue earned from MRPA shows 20% improvement over UNIT_BEST and UNIT_BIG.

Failure Count	Revenue Loss (\$)		# Stopped Jobs		# Unprotected Jobs	
	MLBA	UNIT_BIG	MLBA	UNIT_BIG	MLBA	UNIT_BIG
1	40703	45290	0	0	0	1
2	94077	156016	0	1	4	0
3	207626	285353	0	1	8	10
4	485494	510482	1	7	28	45

Table 2: Protection and Revenue loss with successive Failures

5.4 Backup Allocation

In this set of runs, the primary allocations are given by MRPA. The backups are pre-provisioned using the MLBA and the four heuristics, and the total revenue loss is calculated for each approach, in case of a single failure. In this case, we define the low, medium and high resource availability conditions depending on the value of $\hat{b}f$ in the ranges of (0.08-0.15), (0.2-0.25) and (0.3-0.4) respectively. We report results only for UNIT_BIG as it is found to be the most efficient among the four. To demonstrate the effectiveness of backup sharing, we also add results from the No-sharing case, when no backup resources are shared. Figure-8a shows the total revenue loss for the different algorithms, along with the optimal, at different backup resource availability regions. We show that near-optimality of MLBA as the revenue loss is always within 1.05 times the optimal in medium and high availability regions. In the low availability region, where resource contention is high it can produce solutions that are about 1.02 times the optimal loss. In comparison, UNIT_BIG can be twice as bad as the optimal solution in medium and high availability regions. Figure-8b represents the revenue loss as a percentage of the primary revenue, showing that MLBA achieves between (10-20)% revenue loss in medium to high availability regions, which is around 20% better than UNIT_BIG. By focusing on the revenue loss when $\hat{b}f=0.15$, we show the split of the loss across different jobs in Figure-8c. We observe from the scatter plot in Figure-8d that the loss of MLBA is uniformly lower across small, medium and large jobs.

In Figure-8e, we comment on the scalability of the algorithms as the number of jobs in the system increases, for a constant resource capacity. As the contention for resources increases with more jobs in the system, the MLBA achieves a 18-20% improvement over UNIT_BIG. Finally, in Figure-8f we show the degree of sharing achieved by the backup algorithms. Sharing factor for each resource (> 1 when backup resources are shared), can be estimated by the total backup capacity allocated to the jobs on a resource, divided by the available backup capacity on that resource. Intuitively, higher sharing suggests more efficient utilization of backup resources. We see that in all availability regions, on an average MLBA achieves higher degree of sharing than UNIT_BIG.

5.5 Multiple Failures

Until now, we have focused on the single failure model. Sometimes, the failure of one component can trigger cascaded failures of related components. The notion of shared risk groups (SRG)[5] bundles the resources based on their failure patterns. The failure of one resource instance can trigger the failures of one or more resources in its SRG. Within the QoS-GRAF framework, if primary and backup resources are pre-allocated, such that they are SRG disjoint, then multiple failures limited to a single SRG can be handled. However, the case when multiple un-correlated failures occur in the system, the nature of the problem becomes hard to solve and is beyond the scope of this paper. As a special case, if these failures are not simultaneous, then one way to handle them is to initially provision for a single failure; and then re-run the algorithm at the occurrence of every additional failure. Thus, some re-adjustments are made in the backup allocations at occurrence of every failure. Note that, since with every failure, available capacity in the system decreases, some jobs will eventually have to run without backup protection. If a subsequent failure involves any of the unprotected jobs, then that job will be stopped and its revenue is entirely lost. Table-2 reports the revenue loss, number of unprotected and stopped jobs in a system of 100 jobs with 4 successive, random failures. The MLBA and the UNIT_BIG algorithms are run at the occurrence of every failure. We see that MLBA achieves a (10-20)% lower revenue loss (primary revenue = 1M\$) than UNIT_BIG and also has fewer stopped and unprotected jobs.

6 Related Work

SLA based profit maximization has recently received a lot of attention in the utility computing domain. In [19], authors present an online admission control mechanism for profit maximization of providers. The work uses statistical prediction techniques to estimate remaining service times of jobs. [20] presents an allocation controller that maximizes profits from multiple class SLAs for web servers, when web servers can be added or removed dynamically depending on system load. In the grid utility domain, however, there are additional semantics that need to be considered. For example, a grid application typically is dependent on different types of resources and co-selection requirements can be imposed on these dependencies [18]. Work on SLA based profit maximization has not previously considered co-selection requirements. In addition, grid jobs also require end-to-end reservation of resources in order to provide QoS guarantees even in case of failures. Resource reservation for failure provisioning in networks has been addressed in [16, 17] where network traffic is protected against a single failure with a certain protection guarantee. However, failure provisioning from the perspective of profit maximization has not been addressed. In our work, we consider a failure provisioning model, where the minimum requirements of all jobs are protected. In addition, depending on QoS pricing of multiple level SLAs, we allow a job to be protected at the same level as its primary allocation or at a lower QoS level, enabling finer control for choosing backup allocations and reduced revenue losses.

7 Conclusion

In this paper, we propose QoS-GRAF, a framework for providing revenue maximization in a utility computing grid where jobs have multiple resource dependencies. Multiple class SLAs are specified indicating differentiated QoS pricing. The framework supports advance reservation of primary and backup resources. To solve the revenue maximization problem we propose near-optimal heuristics, MRPA and MLBA that achieve performance within 1–5% of the optimal solution and significantly outperform alternative approaches. The heuristics adjust QoS levels, to achieve revenue earnings across small, medium and large jobs, with efficient resource utilization. We further outline an online backup technique that can handle multiple failures. As part of ongoing work, we are incorporating job penalties explicitly (instead of minimum requirements) in the QoS-GRAF framework. We are also developing backup algorithms for multiple failures that exploit failure history (probability). In the future, we plan to extend QoS-GRAF to incorporate migration cost of failed jobs. Finally, we plan to evaluate QoS-GRAF in a real grid environment.

References

- [1] Gaurav Banga, Peter Druschel, and Jeffrey C. Mogul. Resource containers: A new facility for resource management in server systems. In *Proceedings of OSDI'99*, 1999.
- [2] John Bruno, Eran Gabber, Banu Özden, and Abraham Silberschatz. The eclipse operating system: Providing quality of service via reservation domains. In *Proc. of USENIX '98, New Orleans, LA*, 1998.
- [3] Melissa J. Bucu, Rong N. Chang, Laura Z. Luan, Edward So, Chunqiang Tang, and Christopher Ward. Pem: A framework enabling continual optimization of workflow process executions based upon business value metrics. In *Proceedings of IEEE SCC'05*, 2005.
- [4] Joe DeCarlo and HiPODs Team. *High Performance On Demand Solutions*. www.ibm.com/redbook, 2005.
- [5] S. Dharanikota, R. Jain, Y. Xue, C. Brownmiller, D. Papadimitriou, R. Hartani, G. Bernstein, V. Sharma, and J. Strand. Inter-domain routing with shared risk/resource group(srg). In *Proceedings of MPLS World Congress 02*, 2002.
- [6] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *Proceedings of IWQoS'99*, 1999.
- [7] M. R. Garey and D. S. Johnson. *Computers and Intractability—A Guide to the Theory of NP Completeness*. W. H. Freeman, San Francisco, 1979.
- [8] GRAAP-WG. Grid resource allocation agreement protocol working group in the global grid forum. <http://www.fz-juelich.de/zam/RD/coop/ggf/graap/>.
- [9] Nick Harris, Dale Barrick, Ron Carter, Ron Frantish, M. Guadagno, Ed Gerwill, Steve Mann, A Plu, I. Smith, S. Jamgavkar, and I. Berrios. *Logical Partitions on the IBM PowerPC: A Guide to Working with LPAR on IBM eServer i5 Systems*. www.ibm.com/redbook.

- [10] Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron. Virtual workspaces in grid. In *European Conference on Parallel Computing*, 2005.
- [11] A. Keller, G. Kar, H. Ludwig, A. Dan, and J.L. Hellerstein. Managing dynamic services: A contract based approach to a conceptual architecture. In *Proceedings of NOMS'02*, 2002.
- [12] Dean Kuo and Mark Mckeown. Advance reservation and co-allocation protocol for grid computing. In *e-Science*, pages 164–171, 2005.
- [13] D.A Lifka. The ANL/IBM SP scheduling system. In *Proceedings of the IPPS '95 Workshop Job Scheduling Strategies for Parallel Processing*, 1995.
- [14] H. Ludwig, A. Keller, A. Dan, and R. King. A service level agreement language for dynamic electronic services. In *Proceedings of the 4th IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*, 2002.
- [15] J. M. MacLaren, R. Sakellariou, K. T. Krishnakumar, J. Garibaldi, and D. Ouelhadj. Towards service level agreement based scheduling on the grid. In *14th International Conference on Automated Planning and Scheduling-04*, 2004.
- [16] Cai Ming, Luying Zhou, and Mohan Gurusamy. Dynamic routing of dependable connections with different qop grades in wdm optical networks. In *10th IEEE Symposium on Computers and Communications (ISCC)*, 2005.
- [17] G. Mohan, C. Siva Ram Murthy, and Arun K. Somani. Efficient algorithms for routing dependable connections in wdm optical networks. *IEEE/ACM Trans. Netw.*, 9(5):553–566, 2001.
- [18] Vijay Naik, Chuang Liu, Lingyun Yang, and Jonathan Wagner. On-line resource matching in a heterogeneous grid environment. In *IEEE International Symposium on Cluster Computing and the Grid*, 2005.
- [19] Akshat Verma and Sugata Ghosal. On admission control for profit maximization of networked service providers. In *Proceedings of WWW '03*, New York, NY, USA, 2003. ACM Press.
- [20] Li Zhang and Danilo Ardagna. SLA based profit optimization in autonomic computing systems. In *Proceedings of ICSOC'04*, 2004.