Strategies to Create Platforms for Differentiated Services from Dedicated and Opportunistic Resources*

Shah Asaduzzaman and Muthucumaru Maheswaran†
Advanced Networking Research Lab
School of Computer Science
McGill University
Montreal, QC H3A 2A7, Canada
{asad,maheswar }@cs.mcgil l.c a

Abstract

This paper is proposing a new platform for implementing services in future service oriented architectures. The basic premise of our proposal is that by combining large volume of uncontracted resources with small clusters of dedicated resources, we can dramatically reduce the amount of dedicated resources while the goodput provided by the overall system remains at a high level. This paper presents particular strategies for implementing this idea for a particular class of applications. We performed very detailed simulations on synthetic and real traces to evaluate the performance of the proposed strategies. Our findings on compute-intensive applications show that preemptive reallocation of resources is necessary for assured services. The proposed preemption based scheduling heuristic can significantly improve utilization of the dedicated resources by opportunistically offloading the peak loads on uncontracted resources, while keeping the service quality virtually unaffected.

Keywords: Computing utility, Internet computing, Resource management, Scheduling algorithm, Service oriented architecture

*To appear in Journal of Parallel and Distributed Computing, Elsevier Science

†Corresponding author. Contact address: 3480 University Street, McConnell Engg. Bldg., Room 318. School of Computer Science, McGill University. Montreal, QC H3A 2A7, Canada. Phone: 1(514)398-1465, Fax: 1(514)398-3883, Email: maheswar@cs.mcgill.ca

1 Introduction

Service oriented architecture (SoA) is emerging as a popular paradigm for next generation computing systems. So far implementations of services have mostly relied on clusters of servers or *Internet data centers* (IDCs). These systems are statically dimensioned, which means accurate resource requirement analysis should be done on the services before provisioning them. Accurate requirement estimation is especially a hard problem for services with geographically dispersed users. Static installations with over-provisioning may overcome the problem in some cases, but for services that have variable resource demands, dynamic provisioning is a necessity. With the presence of planetary scale communications enabled by the Internet, different types of resources can be shared among a widespread community for various types of applications. This allows new models for service hosting that bring resources together in hitherto unused ways. One such approach is to consider a hybrid architecture where a large collection of *opportunistically* accessible *public resources* is used in conjunction with small sized clusters of *dedicated resource* that are contracted.

In this paper, we refer to hosting platforms built from a combination of public and dedicated resources as *Public Computing Utilities* (PCUs). Because the dedicated resources are contracted, they are under full control of the PCU *resource manager* and are expected to have reliable and predictable behavior. The public resources are used opportunistically without any contract and are not statically dimensioned. If the public resources come from clients, we can expect the public resource pool to scale up with increasing demand (increasing number of clients). With its hybrid organization, PCU provides several benefits not available with clusters or IDCs. For instance, they provide self-scaling, geographically distributed points-of-presence, increased utilization for dedicated resources, and high compliance levels for service SLAs. Some of these benefits were already evident from our previous study with a restricted model of PCU [1]. In that model, the states of the public resources were considered unobservable and the resource scheduling had to be done in a state-oblivious manner. This approach wasted most of the work done on a public resource if the resource is unable to complete the whole job. Later, we have observed that periodic monitoring of public resources can be enabled without overwhelming the network. In this paper, we present the study of resource management policies in a new system organization that enables status monitoring and failure detection of remote public resources to gainfully engage them within a PCU.

We have studied several approaches to fulfill an application's resource requirements from a PCU and deliver assured services to the clients. Applying our approaches on a compute-intensive application workloads from a research Grid, we show that resource scheduling using dynamic priority based preemptive migrations are necessary to achieve differentiated quality of service for the different clients. We show that these approaches, despite their simplicity, can achieve considerable performance benefits. For example, compared to the heavily provisioned DAS-2 research Grid [2] with a total of 400 CPUs that uses Maui cluster scheduler [3], a model PCU system can be built with only 100 dedicated CPUs, which implies 75% cut in resource provisioning. With opportunistic access to public resources, the same workload as DAS-2 can be carried out in the model PCU with only 12% increase in job execution time.

Although in this paper we consider only processing and network resources as part of a PCU, similar model can be used for a wide range of resource types and can be applied for diverse application scenarios such as license pooling and on-the-fly resource path establishment for data-intensive computations. In Section 2, we provide a simple taxonomy for the application space in SoAs that shows the deployment scenarios where the PCU model is relevant. We also discuss the architectural space of hosting platforms for SoAs to show the relationships between existing paradigms and PCUs. The proposed PCU architecture and its components are described in details in Section 3. In Section 4, we formally define the scheduling problem that is unique to the PCU architecture and discuss scheduling strategies to design on-line heuristic solutions. The relationships among scheduling parameters are analyzed using simplified queueing models. In Section 5, we present the results of simulation based experiments that explore several design parameters of a PCU, as well as the conclusions drawn from the experiments. A performance comparison of a model PCU system with the DAS-2 research grid based on past workload traces is presented at the end of the same section. A brief overview of the related projects is presented in Section 6.

2 Background

To relate the PCU concept with the vast body of existing work on service hosting platforms, we present two simple taxonomies. One examines the architectures for service hosting platforms and the other characterizes the applications interacting with the platform.

Table 1: Characterization of different service hosting platform architectures

Issue	Dedicated Resource Pool	Central Shared Resource Pool		Distributed Shared Resource Pool	
		Small	Large	Deterministic Peering	Opportunistic Peering
Installation Cost	increases with size	low	high	low (harder to peer)	low
Running Cost	depends on size	low	low	low	lowest
Resource Exhaustion	depends on size	definitely	no	could be	no
Distributed Point-of-presence	depends on organization	no	no	yes	yes
Scalable	no	no	yes	yes	yes
Performance Isolation	high	low	high	medium	high-medium
Ease of Use	high	medium	medium	low-medium	low-medium

2.1 Architectural Models

We can classify the service hosting platform architectures based on two aspects of the system: organization and usage. In terms of organization, we can group the systems into centralized and distributed. Similarly, we can group the systems into dedicated and shared based on usage. In Table 1, we have characterized each class of architectures with respect to several design issues. Dedicated resource based architectures are high cost installations of reliable resources and they usually incur either under-utilization or job-overflow depending on the installed resource capacities. Scheduling applications on such resource pools is comparatively easy. Because the resource behavior is deterministic, it is possible to provide very good performance isolation. One common approach to overcome the under-utilization of dedicated platforms is to share the pool of resources among different applications, expecting that their peak loads will not overlap and thus yield more utilization.

The PCU proposed in this paper falls into the distributed shared model. This model can be divided further into two subclasses: with deterministic peering and opportunistic peering. When resources installed in different administrative or geographic domains are shared among the service providers based on some off-line contracts, we call it deterministic peering. In such systems the available capacity of the resources to a particular provider is deterministic. Globus [4] and other Grid like systems may be classified into this group. But it is often very hard to establish such peering among diverse communities due to the need for offline contracts. An alternative method is to opportunistically share the available resources among the providers without any assurance. Computing platforms such as BOINC [5], Condor [6] may be classified in this group. Although this kind of peering is easy to establish, it is really hard for the providers to produce any quality assured service for its user based solely on opportunistic resources. The PCU architecture proposes for a combination of dedicated resource installations and opportunistic peering among distributed installations. Our results show that such a combination can provide better utilization and performance isolation than the individual components.

2.2 Application Models

Several design decisions for a distributed and shared resource based system like PCU are governed by the characteristics of the applications that actually use the system. For hosting purposes, we propose that an application can be characterized along three different axes – *user performance requirement*, *state management*, and *resource requirement* Figure 1 gives a brief taxonomy of distributed applications based on this proposal.

Some of the common user performance requirements include *response time*, *throughput*, and *turnaround times*. Response times are essential in interactive applications such as online games. User requirements for some applications such as media streaming are throughput bounded. In case of real-time applications, the total turnaround time might be a critical issue. The user performance requirements impact the choice of resource scheduling algorithms.

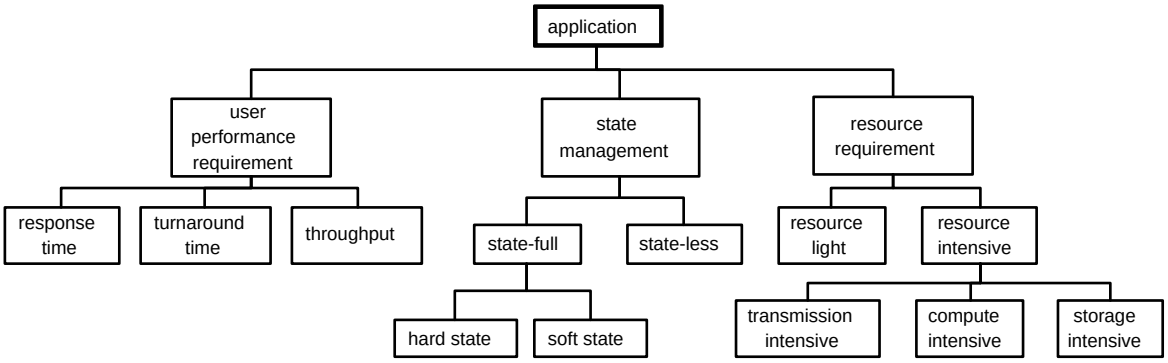

Figure 1: Application taxonomy

Applications can be either *stateless* such as web servers for static documents, or *state-full* such as network game servers. For stateful applications, the volume of state information is an important factor because this affects the amount of time needed for preemptive reallocation of resources. For example, a large weather simulator may initialize a large number of variables in memory and open a number of data files and thus is harder to relocate than a simulator performing a long series of computations on small set of data. The deployment protocol of a stateful application may be based on *soft state* or *hard state*. In case of hard state based system, failure of any node may create a inconsistent system state, whereas soft state deployments can quickly adapt with the failures because they can reconstruct a new consistent state based on available information.

In terms of resource requirements we classify applications as *resource light* (very low resource requirements) and *resource heavy* (very high resource requirements). Resource heavy applications can be further divided into *compute-intensive*, *data-intensive*, and *transmission-intensive* based on the type of resources requested by the application. The PCU service model is generally considered appropriate for resource heavy applications. For an efficient implementation, the platform should carefully manage the resources heavily used by applications. For transmission intensive applications such as media streaming, communication bandwidth is the most critical resource, the resource management strategies of service platforms for such applications need to be implemented by the routers or nodes that play a role in allocating bandwidth on an end-to-end path. For compute intensive applications such as protein folding or molecular model simulators, bandwidth is abundant, but the platform need to carefully allocate the available processing capacity to the applications. Data intensive applications require bulk data storage with a desired degree of reliability. The application needs reliable method of performing transactions on databases and maintaining the consistency of large volume of data.

A PCU system can handle applications with throughput based performance requirements. It is desirable that the applications be either stateless or have low-footprint states. This enables low-overhead migrations when preemptive scheduling scheme is used. Because the PCU system enables access to a large volume of resources, resource-heavy applications naturally fits well here. The PCU architecture explained in this paper is targeted towards compute intensive applications.

3 Proposed System Model and Assumptions

In this section, we describe the model for the different architectural components of the PCU and the assumptions we made on them. We propose a hybrid architecture for PCU, where statically dimensioned clusters of dedicated resources are augmented with opportunistic public resources that are available in large numbers in distributed locations. Although several types of physical and virtual resources can be organized as a PCU, here we consider computing resources only. Several large scale network computing applications including *peer-to-peer* (P2P) file sharing systems such as Gnutella [7], voice conferencing system such as Skype [8] and volunteer computing systems such has SETI@Home [9] have demonstrated the tremendous potential of idle power of desktop computers. The idle capacities of the dedicated resources provisioned by other PCUs may also be used opportunistically during off-peak periods . As in any distributed system, managing this large volume of resources involves three mutually interacting entities - the resources, the resource consumers or the jobs and the resource manager. In the following subsections, we characterize

these components in the PCU and discuss the issues related to them.

3.1 Organization and Characteristics of the Resources

In this paper we consider the scenario of a single PCU provider deploying a single cluster of dedicated resources. These resources are privately provisioned and the PCU provider has full control over these resources. They are highly reliable and their availability extends to almost hundred percent. In addition, we assume that these resources are homogeneous at least within a cluster. The PCU provider has a client base that registers with it in order to get quality assured services. The clients may either purchase the service in exchange of money or redemption of credits earned by donating idle periods of its own resource.

The public resources come from idle capacities of machines owned by others. These may be user desktops across the Internet or unused capacities of the clusters installed by other PCU providers. The major difference between the public resources and the dedicated ones is that the public resources cannot be reserved by the PCU. That is, a job launched by the PCU may be preempted at any time if the resource is required by its owner. Although, in theory, any machine from the whole Internet can be used as a public resource for a PCU, in practice, only a subset of the resources that reside within a given diameter defi ned by network delay and bandwidth constraints will be used. It is possible that multiple PCU providers are close to each other and there is contention for public resources. However, resolving these contentions is a separate issue and a topic of further research. In this paper, we limit our scope to explore the problem of how a single PCU can manage a combination of dedicated and public resources for maximum benefi t of the clients.

In the current Internet, ISPs are ideal candidates for hosting PCUs because they already manage the network access for a large number of clients. In addition, they are ideally positioned to install and manage dedicated clusters of machines for PCUs.

There can be a wide degree of variability in the capacity of the public resources, particularly in two dimensions: *spatial* and *temporal*. In the spatial dimension, the spectrum of benchmarked capacity of the resources spread from very low capacity desktop machines to high-end multiprocessor machines. Existing data regarding the actual distribution of the resources on the Internet is inadequate. To build a model representative of the actual resources, we used the resource capacity statistics from the projects that use the BOINC software [5]. For the purpose of simulation study we derived an empirical distribution from these statistics shown in Figure 2.

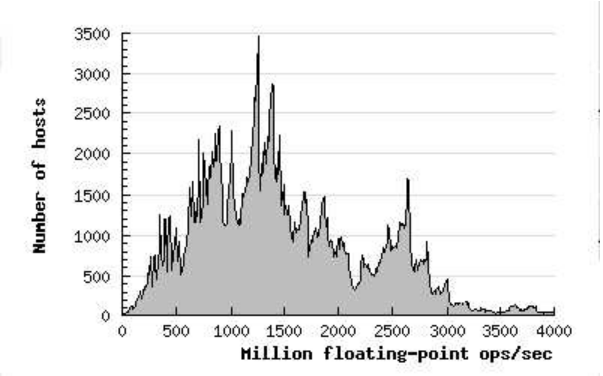

Figure 2: Histogram of frequencies of hosts of different floating point capacities (FLOPS) that are participating in SETI@Home project across the Internet [10]

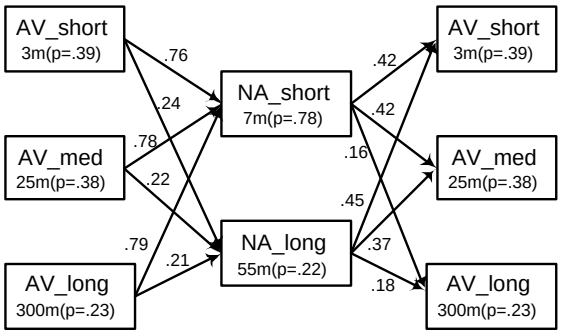

Figure 3: Markov model for host availability characteristics [11]. AV=Available, NA=Not Available. Numbers inside the states are state duration and steady state probability respectively

In the temporal dimension, the availability of the public resources goes on and off according to the user behavior. Several studies have attempted to record the user idleness and machine availability characteristics in different settings [11, 12] and statistically model the distributions [11–14]. The effect of availability on job execution time distribution have been studied in [15]. In addition to user activity, availability is also affected by system software failure, hardware failure and network disruptions. Although different studies have attempted to model particular aspects of failures, a complete model of public resources is not available yet. For our purpose, we modeled the life of a resource as a Markov process that goes through different states of availability according to the transition probabilities and state

durations described in [11]. A resource can fail due to various reasons and we assume that whenever a resource recovers after a failure, it is impossible to retrieve the work done by the job on that resource before failure. However, jobs checkpoint their progress periodically, so the job can be re-instantiated from its last recorded checkpoint.

Although a single physical resource can be time-shared by multiple jobs, we assume that each resource, whether dedicated or public is actually a virtualized unit of resource that executes a single job at a time. It is possible that single physical resource is virtualized into multiple virtual resources, and thus multitasking can occur. The scheduler sees each virtual resource as an independent entity.

3.2 Organization of the Resource Manager

The resource manager of the PCU may be centralized or distributed in architecture. Several issues need to be addressed in developing a full-blown resource manager for the PCU. These include (a) co-allocating different resources such as processing bandwidth, storage capacity, and network bandwidth, (b) using network proximities to derive efficient resource allocations that reduce the loading by PCU on the underlying network and at the same time reduce impact of network congestion on the QoS delivered by the PCU (c) using trust measures of public resources to derive robust resource allocations, and (d) managing the incentives for the participating volunteer resources so that the performance delivered by such resources can be maximized. As a first cut at the problem, we consider the PCU resource manager to manage computing resources only, that are organized in a hybrid architecture as described in Section 3.1. Resource co-allocation [16], location aware resource discovery [17], and trust and incentive management [18] are considered separately and are outside the scope of this paper.

In the centralized organization, the resource manager runs in one of the dedicated resources in the cluster deployed by the PCU provider. PCU *users* (subscribers) submit their jobs to the resource manager for execution and wait for the response. The resource manager is responsible for dispatching all the jobs to appropriate resources, monitoring their progress and, if necessary, rescheduling them on new resources. When a job is spawned on a resource, it is enwrapped in a virtual-machine (or a more lightweight process wrapper) that can measure the resource consumption by the job and communicate the status to the central resource manager. Due to very negligible communication delay within the local network that holds the cluster, the resource manager effectively has all the current state information of the dedicated machines. But the public resources being at significant network distance cannot be monitored closely. Therefore, the resource manager has to rely on periodic status updates from the public resources.

Scheduling and rescheduling being the main responsibility of the resource manager, it may deploy different queuing and prioritizing policies among jobs in order to optimize its service objectives. One of our goals is to design appropriate policies for scheduling and investigate its effectiveness with respect to the proposed architecture. The scheduling policies are non-clairvoyant, i.e., the resource manager does not use any knowledge or estimate of the job execution time in scheduling decisions. However, an estimation of the required resource capacity is expected to be available at job submission time. The user is charged according to the actual resource usage during the execution time of a completed job.

3.3 Users and SLA

PCU can serve a wide range of subscribers or users, coming from both its public resource donors and the outside world, and the users may require the service at different quality levels. The users who require preferential service from the PCU provider, sign SLAs to ensure that service. The SLA for each user specifies four parameters – the maximum workload V_0, service ratio ρ, $0 < \rho \leq 1$, a time window τ to measure the offered load and delivered capacity, and maximum permissible elongation e_{max} for a job.

The parameter V_0 specifies an upper limit on the workload a user can request the PCU to execute, in terms of resource consumption per unit time(e.g., MFLOPS). The PCU is bound to deliver resource capacity at the rate $\rho \min(B, V_0)$, where B is the actual workload requested. If the PCU is not compliant with this rule, it needs to pay penalty or rebate to the user in proportion to the degree of deviation. The user must keep its offered workload B within the maximum workload limit V_0 to be eligible for this rebate. The offered load B is measured continuously over the specified history time window τ. The delivered goodput V (defined below) is also measured over the same time window τ.

Each job submitted to the PCU has a nominal resource consumption rate, C (say, in FLOPS), which implies a nominal execution time T_0 if adequate resource is provided throughout the execution time. However, given that the computational jobs are elastic, the PCU may deviate from delivering this exact capacity and the actual execution time

may be T. We define the quantity $e = \frac{T-T_0}{T_0}$ as *elongation* for the job. The SLA parameter e_{max} is used the classify the completed jobs into two categories. A job counts towards revenue generation only if $e \leq e_{max}$. Otherwise, the PCU provider does not earn anything for that job.

We define the quantity *goodput* V, as the resource capacity delivered to a particular user counting the revenue-generating jobs only. The SLA enforces that $V \geq \rho \times \min(B, V_0)$. If the PCU deviates from this, then it should pay the penalty in proportion to $\max\{0, (\rho \min(B, V_0) - V)\}$. A natural objective of the PCU is to minimize this SLA deviation. So, this quantity is used as one of the metrics to prioritize among the jobs competing for high-throughput dedicated resources.

3.4 Job Characteristics

Although a wide range of applications can take benefit of the hybrid architecture of the PCU, as a first cut, we consider only computational jobs. Computational jobs may have parallel threads, however, we assume them to be independent and do not consider any inter-thread communication. Many of the bag-of-task scientific applications fit in this criteria [19, 20]. Each job is launched in a process wrapper that is able to take a snapshot of the job's progress periodically and store the checkpoint. Also, the wrapper is able to migrate to a newly allocated resource based on instructions from the scheduler. To minimize the down-time, the migration is done in two phases, first the bulk of the process's memory and filesystem footprint is transmitted while the process is live, and then the process is frozen and residual footprint is migrated. This technique has been successfully tested recently with Xen virtual machines [21].

3.5 Migration and Virtual Machines

There are several important concerns about executing jobs on remote machines and migrating them live. One issue is to encapsulate the process so that the interaction between the remote job and host kernel is done in a controlled manner. Besides other techniques, virtual machine can be used for encapsulation. There have been several research works on efficient implementation of virtual machines and almost local equivalent performance is now achievable.

Although virtual machine is the best option for secure isolation of the guest processes, one problem is that a full-blown virtual machine is too heavyweight for migration. Considering this fact, several research projects have developed efficient process migration techniques that can successfully live-migrate server applications [22]. However, virtual machine based techniques are gaining popularity because of the residual dependency [23] and infrastructure incompatibility [24] problems of process migration. Virtual machines can now be live-migrated that greatly reduces the actual down-time of the applications. Recently, heavily loaded web severs and game severs running on Xen has been live migrated with down time in the order of 100ms, although the total migration time spans over $2-3$ minutes [21]. Still we believe, to enable deployment of PCU, further research need to be done on virtual machine techniques to achieve lower memory footprint alongside guest process isolation. Such lightweight techniques are also important for the widespread availability of virtual machines on end-user desktops. With the introduction of hardware assisted virtualization mechanisms in the mainstream processors such as Intel [25] and incorporation of VM monitors into the common desktop OS kernels [26], widespread availability of VM enabled desktop machines can be expected in near future.

4 Design of the Scheduling Policies

Using the PCU model we developed in the previous section, here we explore different design considerations to devise an effective scheduling algorithm for the PCU resource manager.

4.1 Defining the Problem

Having a stream of requests from different users, the prime task of the resource manager is to allocate one resource for each of the jobs. Given the finite number of high capacity dedicated resources, not all the jobs may be entitled to a dedicated resource immediately. If there was no access to any resource other than the dedicated ones, it would be a *n*-server single queue scheduling problem. Any solution to that would apply some priority rule on the queue and schedule the most prior jobs in one of the dedicated resources. The other jobs either remain idle in the queue or are dropped. There are well studied on-line heuristics for this problem such as EASY Back-Fill [27] or Maui [3]. The

optimization goal could vary depending on the type of applications, but most of the schedulers try to minimize the completion time of the jobs.

Because we have access to a large number of public resources on the Internet, low priority jobs can get some progress running on these resources instead of sitting idle in the queue or getting dropped. Given the wide variation in the performance of public resources, this progress is quite unpredictable. To maintain the order of priority, it may be necessary to migrate the jobs running on public resources back to dedicated cluster as soon as some resource becomes free. However, too many migrations will eventually swamp the network and the down-time for preemptive migrations will grow. The scheduling priorities are to be designed carefully to prevent unnecessary migrations.

Due to heterogeneous public resources, the scheduling problem becomes harder and analytically intractable. In addition to the standard goal of minimizing the response time for each job, the schedule needs to consider the SLAs contracted with each user and attempt to minimize the penalty paid by the PCU due to any deviation. Another goal is to minimize the network overhead to keep the delays in migrations within acceptable limits. Keeping these different goals in mind, we attempt to calibrate the design parameters of the resource manager and study their impact on performance.

4.2 Preemptive Migrations

In the absence of any SLA with the users, it would be sufficient to maximize the throughput of jobs in order to bring higher revenue for the PCU. This can be achieved by maximizing the utilization of dedicated resources disregarding priorities of the jobs. A simple algorithm to achieve this is to schedule all the high priority jobs in dedicated resources as long as one is available, and schedule the rest of the jobs on public resources. Since the number of public resources is virtually infinite, there will always be enough resources to schedule the jobs in queue. This scheme would certainly be unfair to long-running jobs that happen to arrive when all the dedicated resources are allocated. Such jobs have their fate stuck with some public resource for life.

In order to consider fairness among jobs in terms of access to the dedicated resources, a straightforward approach would be to allocate the dedicated resources to all the running jobs in round-robin fashion. Although this scheme will ensure uniform progress for all the jobs, the high number of migrations will cause severe congestion in the link that connects the cluster to the Internet. One better idea is to dynamically and periodically assign priorities to all the running jobs and make sure that higher priority jobs migrate to the dedicated resources. Then the problem becomes computing the priorities intelligently, such that it does not incur high number of migrations whereas it ensures proportional share of dedicated resources according to the SLA.

4.3 Priority Functions

Given that a priority function for controlling preemptive migrations is central to the scheduling algorithm, we try to devise a proper function now. The idea is to assign priorities to each job such that lagging or deprived jobs get higher priority to get scheduled on dedicated resource. Two primary objectives considered in computing the priority are – a) minimize the completion time for each job b) minimize the deviation from SLA for each user. These priorities should be computed from the available information about the jobs. We assume that the resource manager can maintain a job table with entries for all running jobs in the system. Now, based on current level of progress, the requested resource capacity, and the total amount of time elapsed from arrival, the resource manager can compute a relative elongation for all the running jobs. The resource manager also maintains a user table to keep track of current level of SLA deviation for each user. Then a linear combination of the deviation and the elongation values can be used as a priority for each jobs. For our simulations we simply added these two values. There can be other contributors to the priority value. Jobs that are delayed too much and do not have any chance to complete within the SLA defined elongation margin (in order to be counted in the goodput), are de-prioritized by forcing very negative priority values.

In the dynamic preemptive migration algorithm, the priorities are recomputed every epoch and a new ordering of the job table is computed. Then the N_p highest priority jobs are mapped on dedicated resources, where N_p is the total number of dedicated resources. Note that a major fraction of these N_p jobs may already be running on dedicated resources, so they need not get rescheduled. Also, the algorithm prevents rescheduling of the jobs that are in migration or recently migrated. Thus, number of migrations every epoch is much smaller than N_p. Algorithm 1 shows a skeleton for the scheduling algorithm. Computing the priorities and reordering the table is not computationally expensive because, we don't need a complete ordering of all the n running jobs in the job table. Selecting top N_p entries will be sufficient and this can be done efficiently in $O(N_p \log(n))$ time (N_p is a constant here) if a heap data structure for jobs entries are maintained.

Algorithm 1 Skeleton scheduler

```
1:  for Every scheduling epoch do
2:    while jobs in InputQueue do
3:      j = first job in InputQueue
4:      if j is too old then
5:        drop(j)
6:      else if Free dedicated resource found then
7:        Schedule j in dedicated resource
8:      else
9:        Query for k public resource {k is the replication factor}
10:       if 0 < i ≤ k free public resources found then
11:         Spawn i replicas of j on public resources
12:       else
13:         break
14:       end if
15:     end if
16:   end while
17:   Based on life-pulses, detect failure of all remotely running jobs
18:   Compute priority of all the N_r jobs currently running
19:   if Among most prior N_p {number of dedicated resources} jobs m are running on public
      resources then
20:     Start migrating the forerunner replica of these m jobs to dedicated resource pool
21:     There will be m among the remaining N_r − N_p jobs that are running on dedicated
        resource. Start migrating them to public resources
22:   end if
23:   for All jobs j running in public resources and not migrating do
24:     If not all k replicas are up, replenish the replicas
25:   end for
26: end for
```

4.4 Updating Remote States and Failure Detection

Once a job is scheduled on a public resource, it is important to get the updated information on resource consumption by that job. We assumed that each job running on a public resource sends a small progress report (not a complete checkpoint) to the resource manager at intervals equal to (but not necessarily synchronized to) scheduling epoch. The resource manager updates its job table using this information. Also, these reports act as life-pulses for the remote jobs, because after a timeout period of no progress reports, the resource manager can assume that the remote job has failed.

4.5 Replication

In order to mask the unreliability of the public resources, we have used multiple replicas of a single job running in parallel on multiple public resources. On detection of failure of any one of the k different replicas, the resource manager re-spawns another replica on a new resource, and thus it always tries to maintain k replicas alive. The idea is to minimize down time due to failure to almost zero, by maintaining at least one replica alive all the time. In case all the k replicas fail at the same time, the resource manager has to re-spawn them from the last recorded checkpoint. Note that jobs running on public resources do not checkpoint their progress periodically, whereas the ones inside the cluster does. This is because the only reliable place to store the checkpoint is inside the cluster, and storing periodic checkpoint from remote jobs would cause huge amount of traffic load on the bottleneck link of the cluster. An estimate of appropriate value for k can be found from the analysis given in the next section.

4.6 Analyzing the Design Parameters

The parameters of the scheduling scheme discussed above are interrelated along with other system parameters such as size of the dedicated cluster and workload. Understanding these relationships is important for an optimal tuning of

these parameters. Due to its complex nature, a realistic model of the system is hard to solve analytically. Nevertheless, it is useful to resolve the dependencies under simplifying assumptions. Here we derive some approximate relationships among the parameters using some simplified queuing models.

Total capacity of the system

Let N_d, N_p be the number and T_d, T_p be the mean throughput of dedicated and public resources respectively. Let P_{av} be the steady state availability of a public resource and r be the number of replica per job. Assuming zero cost of migration the resource collection provided by the PCU is equivalent to N dedicated resources, where $N = N_d + \frac{T_p}{rT_d} P_{av} N_p$. If we feed an Erlang's loss system of capacity N with unit-workload jobs (1 FLOP) in a Poisson process at mean rate (λ) equal to the workload arrival rate of PCU, we get an estimate of maximum deliverable throughput of the PCU applying Erlang's loss formula [28] on the equivalent system -

$$\text{Throughput} = \lambda \left(1 - \frac{(\lambda/\mu)^N}{N! \sum_{i=1}^{N} \frac{(\lambda/\mu)^i}{i!}} \right)$$

The bottleneck network link

To analyze how the bottleneck link capacity restricts the number of dedicated resources and also the incoming load, let us assume N_d be the total number of dedicated resource, B_{up} (bps) be the uplink bandwidth of the bottleneck link that carries data from the dedicated cluster to the public resources and B_{down} be the downlink bandwidth. Let λ be the arrival rate of jobs and L be the mean execution time of each job, if executed uninterrupted in a dedicated resource. In light load, when $\lambda L < N_d$, all the jobs will be served by dedicated resources, so there will not be any significant amount of transmission through the bottleneck link. At high load, when $\lambda L \gg N_d$, most of the incoming jobs will be initially spawned on a public resource, and as a result there will be significant number of migrations every epoch. In the worst case, all the incoming jobs go to public resources. So, the transmission load on bottleneck due to new jobs,

$$\lambda_s = km\lambda$$

where, m is the mean size of the memory footprint of a job and k is the replication factor. For the load due to preemptive migrations, there can be N_d inbound and N_d outbound migrations every scheduling epoch, in the worst case. If the epoch length is δ, total transmission load on the outbound link,

$$\lambda_b = km\lambda + \frac{km}{\delta} N_d$$

For sustained steady state transmission, this load must be less than the capacity, i.e.,

$$\begin{aligned} \lambda_b &< B_{up} \\ \Rightarrow \lambda &< \frac{B_{up}}{km} - \frac{N_d}{\delta} \end{aligned}$$

Here, both N_d and δ are adjustable parameters and the arrival rate λ can be restricted for particular values of N_d and δ. For example, $N_d = 100$, $\delta = 120$sec, $k = 2$, $m = 1$megabytes, $B_{up} = 100$Mbps restricts maximum arrival rate to 5.4jobs/sec. Similarly, the downlink bandwidth B_{down} restricts the transmission load such that,

$$\begin{aligned} \frac{mN_d}{\delta} &< B_{down} \\ \Rightarrow N_d &< m\delta B_{down} \end{aligned}$$

Length of scheduling epochs

Regarding the length of the scheduling epoch, we already mentioned that large epochs leave failures of public resources undetected for a long durations. On average, each failure takes $\frac{\delta}{2}$ seconds to be noticed and one migration time $(= T$ seconds$)$ to be respawned. So, total job down-time for each failure is $T + \frac{\delta}{2}$. Such down-time occurs for a job only when all the replicas of the job running on the public resources fail concurrently. Say, among all the jobs running

on public resources, rate of such concurrent failure is y times per second. Assuming both the failure and respawn processes to be Poisson processes, the steady state number of jobs down due to failure (n_f) can be derived using the $M/M/1$ queuing model [29] –

$$n_f = \frac{y}{\frac{1}{T+\delta/2} - y} = \frac{y\delta + 2Ty}{2 - (y\delta + 2Ty)}$$

Using similar approach, we can derive the steady-state down-time of jobs due to preemptive migrations. If, x migrations are initiated every epoch ($x \leq N_d$) and each migration takes T seconds to complete, then steady state number of jobs down due to migration,

$$n_m = \frac{x/\delta}{1/T - x/\delta} = \frac{xT}{\delta - xT}$$

If we want to minimize overall downtimes of the jobs due to both failure and migration, we need to find δ that minimizes,

$$f(\delta) = n_f + n_m = \frac{y\delta + 2Ty}{2 - (y\delta + 2Ty)} + \frac{xT}{\delta - xT}$$

For example, for $x = N_d = 100$, $T = 100$ sec, $y = 1/200$ (1 failure per 200 seconds), $f(\delta)$ is minimized for $\delta = 180$ seconds.

Number of replicas

It is beneficial to run replicated copies of the jobs on different resources, when they are scheduled on unreliable public machines. From the reliability point of view, the higher the replication factor k is, the better the chance of survival. But higher replication is definitely expensive and wasteful in terms of resources. Also, increasing k would increase the bottleneck network load for checkpointing and migrations. A desired value of k is the one that allows at least 1 replica always up when running on public resources.

Say a job is replicated on k public resources. If we assume that the failure process is memoryless, each resource i has an exponential random runtime before failure with mean time $1/\lambda_i$. Let us assume the mean recovery or migration time is $1/\mu_i$ and the distribution is exponential. Also failure and recovery of all resources are independent.

For any resource i, its expected uptime $E(up) = 1/\lambda_i$ and expected downtime $E(down) = 1/\mu_i$. According to renewal theory [29], probability that it is available at any given time is:

$$A_i = \frac{E(up)}{E(up) + E(down)} = \frac{1/\lambda_i}{1/\lambda_i + 1/\mu_i} = \frac{\mu_i}{\mu_i + \lambda_i}$$

If all the resources are identical, in that case, say $\forall i A_i = p$. Now, if X is the available number of resources at any given time, then

$$P(X = j) = \binom{k}{j} p^j (1-p)^{k-j}$$

Therefore,

$$P(X \geq 1) = 1 - P(X < 0) = 1 - (1-p)^k$$

$$\Rightarrow k = \frac{log(1 - P(X \geq 1))}{log(1-p)} = \frac{log(1 - P(X \geq 1))}{log(\lambda) - log(\lambda + \mu)}$$

So, if we want, with 95% probability, that 1 resource be available at any time, and we have MTTF $\frac{1}{\lambda} = 5$ minutes and recovery time $\frac{1}{\mu} = 2$ minutes, then,

$$k = \frac{log(1 - 0.95)}{log(1/5) - log(1/5 + 1/2)} = 2.39$$

5 Simulation Study

We performed three sets of experiments on simulated PCU systems as described in the following sub-sections. The first set of experiments described in Section 5.3 weigh between different scheduling strategies. Having a chosen

strategy, the next set of experiments, described in Section 5.4, explore the parameter space of the system to investigate different trade-offs and optimal settings. A set of design principles that is drawn based on the simulation results and the analysis given in Section 4.6, is presented in the following sub-section 5.5. Last, in Section 5.6, we evaluate a model PCU system with DAS-2 Grid using an actual computational workload trace. Model parameters for the simulated systems and workload data sources used for different experiments are described in sub-section 5.1 and 5.2, respectively.

5.1 Simulation Model

We developed a discrete event simulation model using Parsec [30] for a PCU system. The dedicated cluster was assumed to have up to 200 Pentium-IV, 3.2GHz machines connected by a Gigabit Ethernet. The benchmark capacity of each Pentium-IV machine was 3.5GFlops. The number of public resources was large compared to the size of the dedicated cluster. We assume that the number of machines reachable from the cluster within acceptable network delay and bandwidth constraints is $10,000$ (for comparison, the total number of participating hosts in a popular BOINC project is in the order of $100,000$ [10]). The network parameters were set to 100ms of latency and 1Mbps bandwidth. This is reasonable if we harvest desktops from home users because they connect to the Internet using 1Mbps or better DSL or cable modem, and the round trip times between machines from North America and Australia is in the range $150-250$ms. The public resources are distributed widely across the Internet where the backbone capacity is much higher than the endpoint link. So the background traffic on the backbone will not significantly perturb this minimal bandwidth. The distribution of computational capacity of these public resources are empirically derived (Figure 2) from the distribution of capacities of the participant hosts of the SETI@Home project [10]. The failure model of the public resources are modeled as Markov process schematically described in Figure 3 and it corresponds to the analysis given in [11]. The cluster is connected to rest of the Internet through one or more edge routers which creates a bottleneck for traffic in and out of the cluster. We assume the total capacity of the bottleneck link to be 100Mbps.

5.2 Workload Data Source

We used three different workloads for different sets of experiments, that is appropriate for the particular experiment objectives. To test the performance of the chosen algorithm on a real workload we chose the workload trace from DAS2 5-cluster research Grid over year 2003 that was presented in [31]. DAS2 is a Netherlands based academic research Grid, made up of 5 clusters of Pentium-III machines spread across 5 universities. Four of the five clusters consist of 64 P-III CPUs each and one has 144 CPUs. We took the run-time information for each job multiplied by a standard benchmark of a P-III 1GHz CPU (1.5GFlops) as the workload for each of the job. Also, the trace has the information on average amount of memory allocated to each jobs, so we took it as the memory footprint size. When a job is migrated, some of the information stored in the filesystem also needs to be carried. We assumed that total amount of data that needs to be carried for each migration is 3 times the average allocated memory for the job.

To evaluate the correlation of per user goodput and SLA capacity obtained from different algorithms, we needed a synthetic workload model where we can control the job arrival rate, but the arrival pattern and job execution-time distribution are modeled after real workload traces. For this purpose we chose the workload model defined by Lublin et.al. in [32]. To explore the sensitivity of the model over a range of design parameter values, we needed more control and visibility of the workload, so we used a Poisson workload model with varying arrival rates. Each of the jobs had an workload of 5000 CPU-seconds on a dedicated machine, and had 1MB of memory footprint.

5.3 Choosing Between Scheduling Schemes

Among different service objectives of the PCU are minimizing the down-time and completion time of the jobs, and enforcing the proportional share of resources according to the SLAs. Replication has been used in order to minimize down-times, and the priority function is carefully designed to minimize the job running time as well as establishing fairness. One major question remains whether we need preemptive migrations to obtain service differentiation, such that the resource capacities delivered to the users are proportional to their SLA defined values. We have compared preemptive and non-preemptive strategies in terms of achieving this service differentiation. To quantify this achievement for different algorithms, we measured the correlation of per user goodput (V) and the SLA maximum allowable workload parameter V_0 (defined in Section 3.3).

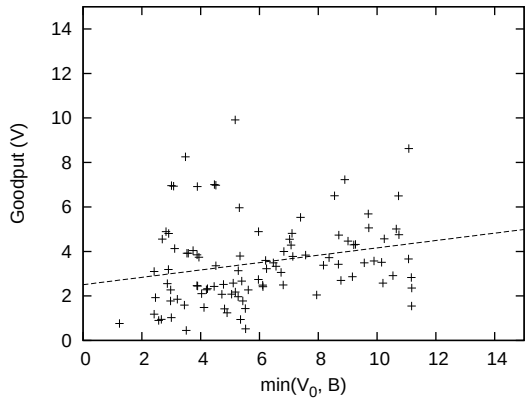

Figure 4: Correlation of Goodput with SLA MaxLoad parameter: Static Allocation. Pearson's correlation coefficient, $r = 0.1655306734$

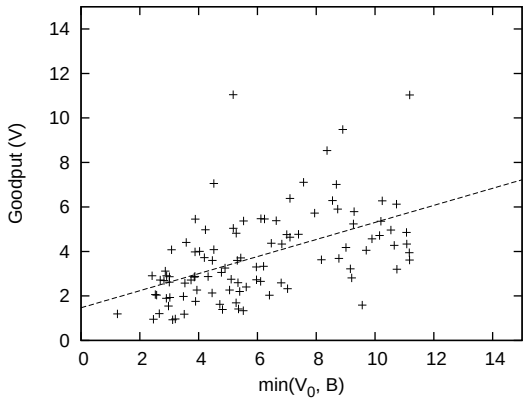

Figure 5: Correlation of Goodput with SLA MaxLoad parameter: Dynamic Priority Preemption. Pearson's correlation coefficient, $r = 0.3835632358$

In the experiment, we fed the PCU with synthetic workload from 100 different users. The V_{0i} (MaxLoad) parameter of the SLA for each user i was chosen randomly. The PCU had 100 dedicated resources, and the total concurrent load on the PCU was on average 5 times higher than the dedicated resource capacity. The workload offered by each user (B_i) was uniform across users and uncorrelated to their V_{0i} parameters. To generate the workload from each user we followed the model described in [32]. Among the two algorithms we compared, the first one (Non-preemptive) assigns the dedicated resources statically to the jobs at arrival time, considering the job from the user with highest SLA deviation (as defined in Section 3.3) first. Remaining jobs are assigned on available public resources. During the course of execution, a job is never migrated preemptively between dedicated and public resource pools. But, as public resources may fail, replicas of the jobs are reinstantiated on new public resources. In the second algorithm (Priority preemption), instead of allocating the resources statically, the scheduler dynamically computed the priority of each job as described in Section 4.3, and migrated the most starving jobs of the most starving users into the dedicated resources.

Figures 4 and 5 show the scatter-plot of user goodput (V) vs $\min(V_0, B)$ for the two algorithms, respectively. We observe that goodput have much higher correlation when preemptive migration with dynamic priorities is used. Although preemptive migration has high cost (network load and delay) associated with it, this result suggests that we need it in order to achieve service differentiation, especially if the workload has a substantial number of long-running jobs. This happens because the SLA deviation of different users change during the runtime of long running jobs and the best way to deal with this is to dynamically allocate the scarce dedicated resources. Motivated by these results we have chosen the Priority Preemption scheduling for further performance analysis.

5.4 Setting the Design Parameters

In this section, we explore different performance metrics for the chosen Priority Preemption scheduling strategy and the associated trade-offs. In order to focus on the global performance of the PCU, we kept the workload generated by different users similar in this set of experiments, and SLA parameters were also set to be same for all users.

First we set up the experiments to measure the mean overall throughput of the PCU using this algorithm, compared to the theoretically achievable maximum throughput from the same system assuming zero cost for migration. An estimation of the total capacity and maximum throughput of the system can be derived from the analysis in Section 4.6.

Figure 6 shows the mean overall throughput from PCU for a range of load levels and comparable theoretically maximum achievable throughput of equivalent systems. The above figure also plots the maximum achievable throughput from a system with only N_d dedicated resources. We observe that despite down-time for preemptive migrations, at moderate loads, PCU can deliver throughput almost equal to the ideal system that ignores migration cost. At high load however, the throughput marginally decreases, due to excessive network overheads for migrations.

Now we examine the effect of increasing the size of the dedicated cluster. One obvious effect of this is an increased capacity of the PCU. But increasing the number of dedicated resources increases the possible number of migrations per scheduling epoch and hence higher migration cost, both in terms of job execution time and network overhead. Figure 7 illustrates the overall goodput of the PCU on the job arrival load vs dedicated cluster size space. It is important to note

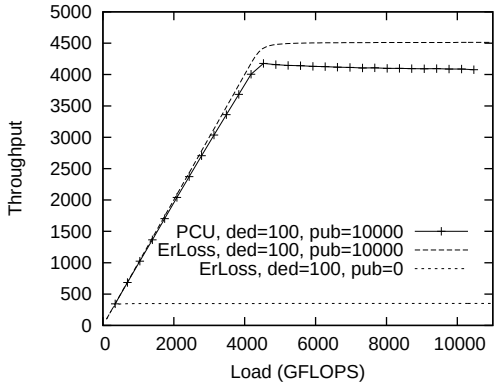

Figure 6: Comparing PCU throughput at different loads with Erlang Loss systems with equivalent number of resources. $N_d = 100$, $N_p = 10000$, $r = 2$, $P_{av} = 0.7$, $\frac{T_p}{T_d} = 0.34$

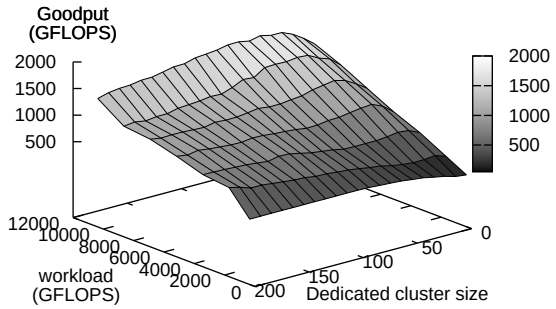

Figure 7: How Goodput is affected by number of dedicated resources at different loads

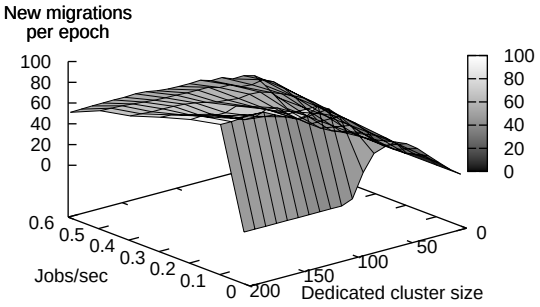

Figure 8: Load on the bottleneck network link due to preemptive migrations

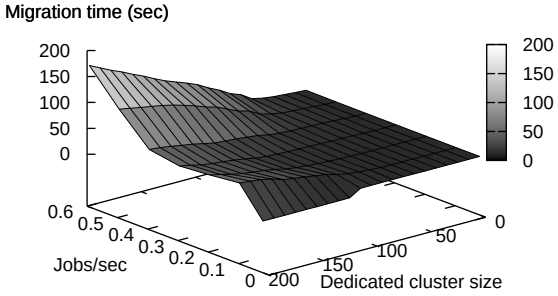

Figure 9: Time required for a migration through the bottleneck link

that network load increases with the number of migration decisions every epoch, and after some limit, the migration time grows towards infi nity because the queues in the bottleneck links gets saturated.

Figures 8 and 9 show the effect of migration on the network. Figure 8 plots the number of new preemptive migrations started every epoch and Figure 9 plots the average time taken for each migration, both on the job arrival load vs cluster size space. We observe that network load increase with increased job arrival load and also with increased size of dedicated cluster. Beyond a certain level of network load, the bottleneck link and routers on them gets saturated and each migration takes exponentially longer time to complete. This rise in network load limits the maximal goodput and minimal elongation across a certain band of the arrival load and cluster-size range. However, since the scheduler does not preempt the already ongoing migrations, the increasing network load actually reduces the number of new migrations every epoch acting as a negative feedback. Accordingly, goodput or elongation does not change much at higher load and bigger cluster size.

With the results of above experiments in hand we can now choose the dedicated cluster size and the load on the system from the safe regions of the graphs. As an example settings, we did the latter experiments with 100 dedicated resources and at a moderate workload that is 5 to 10 times higher than the total capacity of the dedicated cluster.

The next parameter we study is the number of replicas that concurrently run when a job is scheduled on public pool. The main rationale behind replication is to mask the failures of public resources and reduce the down-time of the application due to failures. Here we observe the effect of replication on job's down-time and elongation, using the more versatile simulation model. General observation is that running only 2 concurrent replicas, dramatically reduces

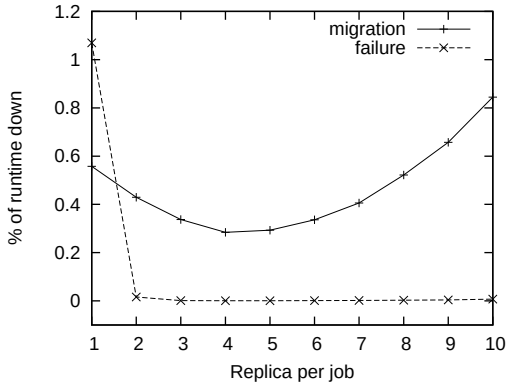

Figure 10: Downtime due to failure and preemptive migration, for increasing degree of replication

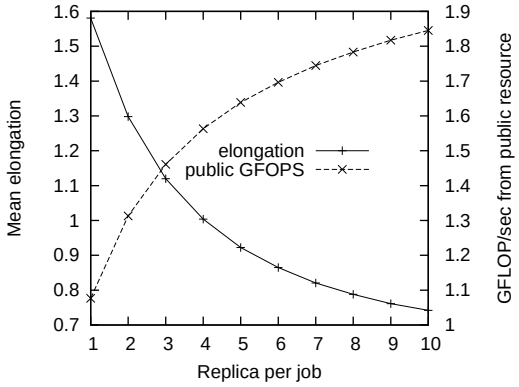

Figure 11: Higher degree of replication reduces elongation, because it increases the chance of getting higher capacity public resource

the chance of concurrent failures and hence reduces the down-time (Figure 10). On the other hand, higher degree of replication increases the network load and therefore, after some point, each migration takes longer to complete and job down-time for migrations actually increases. Another interesting effect of replication is that when we run many replicas on different public resources in parallel, it increases the chance of getting a higher throughput public resource from the global pool, and therefore, runtime of the jobs keeps decreasing (lower elongation) as we increase the degree of replication (Figure 11). For the same reason, the need for preemptive migration decreases with higher degree of replication, and we observe a decrease in job down-time caused my migrations, with the first few increases in the degree of replication (Figure 10). Although we gain in performance from public resource with increasing replication, it would greatly increase the traffic on the global network and also apparent waste of public resources. Therefore, we suggest that replication beyond order of two may not be desirable. A theoretical estimation of this replication factor is provided in Section 4.6.

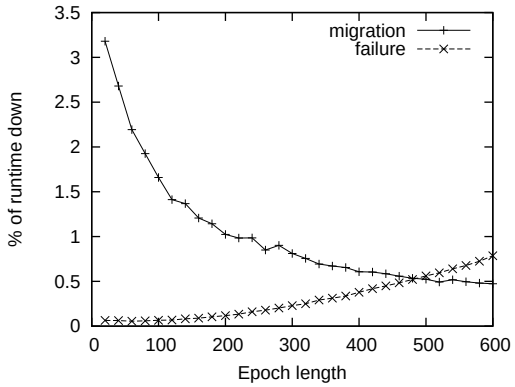

Figure 12: Downtime due to failure and preemptive migration at different lengths of scheduling epoch

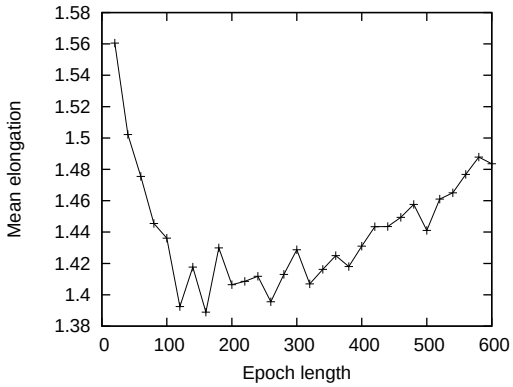

Figure 13: How elongation is affected by the length of the scheduling epoch

Another important design parameter is the length of the scheduling epoch i.e., the elapsed time between scheduler invocations. Too short epochs implies very frequent migration decisions and hence, high network load. But as the epoch length increases, the time to detect a failure also increases, because failure is detected by the scheduler from timeouts of the life-pulses from remotely running jobs and this evaluation is done only when the scheduler is invoked. This adversely affects the down time due to failure, as shown in Figure 12. Figure 13 shows the mean elongation for increasing length of the epoch. We can observe that there is a optimal epochs length for this particular system settings.

5.5 Design Guidelines

Following the observations from both the simulation experiments and the queueing analysis presented in Section 4.6, a set of design principles can be outlined for future design of PCU systems –

- preemptive migration is useful to enforce service differentiation although it incurs overhead.
- The number of dedicated resources that can be deployed in a single cluster should be limited, depending on the capacity of the network link that connects it to the Internet. For larger deployments (e.g. more than 100 machines), multiple clusters need to be setup in different network domains.
- It is useful to keep concurrent replicas of a job, but with more than two replicas, the overheads outweigh the benefits.
- The scheduling epoch should be in range of 1-2 minutes, either longer or shorter epoch length causes performance degradation.
- PCU can tolerate workload much beyond the total capacity of the deployed dedicated resources and hence, PCU provider can oversubscribe users to a certain extent. An estimate of the reachable public resources within certain delay and bandwidth constraints and their failure rates can be used to determine the maximum limit of such overbooking.

5.6 Performance Comparison with Grid Systems

Once we got the acceptable ranges for all parameters, we show how the scheduler works with the Grid workload trace from DAS2 [2]. The main argument behind a PCU-like hybrid architecture is that dynamic provisioning of opportunistic public resources allows much higher volume of workload to be handled by the PCU than having a system of statically provisioned dedicated resources only. Alternatively, offloading much of the work to the public resources, the the volume of dedicated resource provisioning can be cut down by a major fraction without affecting the performance of the service significantly. Both of these facts result in a much higher utilization of the expensive dedicated resources. In our simulation experiments, the workload from the DAS2 trace has been fed to a model PCU system with a range of dedicated resource volumes. From the results of the simulation plotted in Figure 14 we observe that a PCU having dedicated resource volume as low as 25% of the DAS2 resources, yields dramatic increase in utilization of dedicated resource as high as 250% with not more than 10% increase in job runtime.

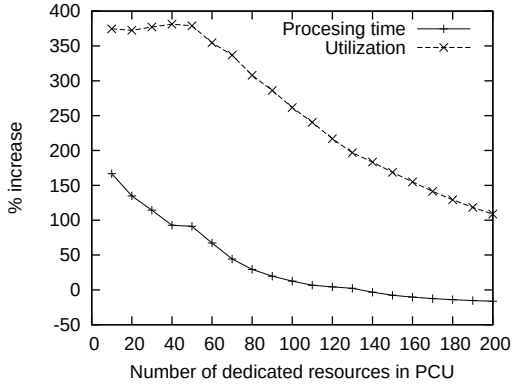

Figure 14: Comparing PCU with DAS2 Grid: gain in resource utilization and increase in running time

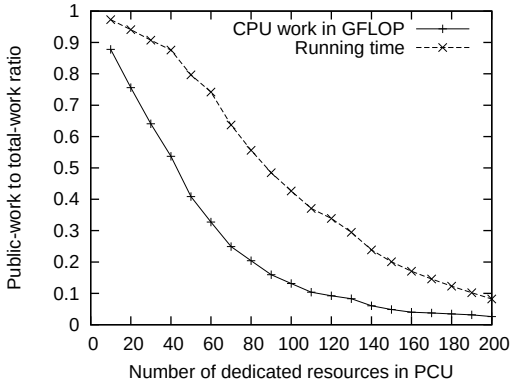

Figure 15: Amount of work done on public resources

The opportunistic resources become a major contributor to the total computational work, especially in times of peak loads, despite their low trust and high failure probabilities. We can observe in Figure 15 that for a reasonably sized PCU, approximately half of the total computation (30% to 60%) is done on the opportunistic resources. If we look into the down-time of the jobs due to migration and failure the general observation is that the although the downtime for migration time is higher than downtime for failure, it is insignificant compared to the gain in the service capacity we achieve.

6 Related work

Platforms that can be used to implement services can be broadly classified into three categories – i) systems based on dedicated resources only, ii) systems built on opportunistic public resources, iii) hybrid systems that combine dedicated and public resources for QoS assurance. First, we explore the literature that addresses the problems related to these three classes of architectures. Then, we review techniques found in existing literature for QoS assurance and live migration of applications.

Dedicated resource based systems

The dedicated resource based systems are mostly homogeneous and deterministic in nature. Scheduling on a fixed size deterministic pool of resources is a well studied problem both in computer science and operations research. Although several optimal algorithms are available [33] for simpler scheduling problems, most of the interesting and practical scheduling problems are computationally intractable. When preemption is possible, there are optimal polynomial time algorithms for scheduling jobs with arrival time and due date constraints on a single processor [34]. Also for the two processor case arbitrary jobs with certain precedence constraints can be scheduled in polynomial time [35]. However, scheduling jobs with arrival time and deadline constraints is proven to be a NP-hard problem for more than two processors [36]. In fact [37] proved that optimal scheduling of jobs in multiple processors is impossible if any of the 3 parameters – arrival time, execution time or deadline is unknown. Because in an online scheduling scenario, resource allocations have to be carried out with incomplete information regarding jobs, heuristic solutions are appropriate for this situation. A good survey of online scheduling heuristics can be found in [38].

Public resource based systems

Several research projects target to harvest idle capacity from public resources such as Condor [6], BOINC [5, 39], OurGrid [20] and Cluster Computing on the Fly [19]. The Condor project is one of the studies to investigate the availability pattern of the workstations in a university computing facilities [11]. Later they developed mechanisms for advertising and discovering matching resources [40] in a Condor like environment. The BOINC project has developed tools for migrating chunks of parameter datasets for parameter sweep applications to take benefit of idle public resources. OurGrid project has tried to apply Grid computing concepts [4] for constructing Grid systems based of public resources. Cluster computing on the fly project has introduced the notion of wave scheduling [41] in order to take advantage of the variation of idle time based on geographic time zones. But none of these projects have considered the case of having hybrid resource pools, to produce commercially usable assured services.

Hybrid systems

In [42], Kenyon showed that it is theoretically possible to make significant improvement in QoS guarantees for the application using public resources if a pool of dedicated resources is available. In other words, computing clusters can provision minimal resources to handle the average load and delegate the jobs to public resources in times of peak load, but still complying with the SLAs for their clients. In [1] the authors proposed scheduling heuristics for such settings and compared their performance with other classical scheduling algorithms. In order to keep the network overhead minimal, the scheduler was designed to work being totally oblivious of the global system state. Especially, there was no information available on the status of a job running on public resources. Neither was there any checkpointing to facilitate migration of partially completed jobs and hence, rescheduling of jobs implied restart. Due to this oblivious nature of the system, there was a significant loss of the work done on public resources. One major unresolved question was how the scheduler can benefit if some periodic status update is available and what the impact on network.

In this work we have enabled periodic update of status information from the remote jobs, that helps the scheduler to detect failures early and also to decide about preemptive migration for the benefit of the lagging jobs. We have tried to engineer the scheduling policy and the system parameters such that the additional information can be utilized without incurring significant overhead. The trade-offs between design options are presented and usable range of operating points are suggested.

QoS assurance in distributed systems

One major goal of the *Resource Management System* (RMS) of a PCU is to maintain QoS according to the SLAs signed up with its clients. Architectures of SLA compliant resource management for both centralized and distributed pool of dedicated resources has been studied in several research projects such as Oceano [43] and Globus Grid [4] [44]. However, study of scheduling algorithms with detailed performance evaluations were not carried in the above works. Performance evaluation of scheduling heuristics for cluster based hosting centers are found in [45] [46] [47] with different optimization goals in different cases. But creating assured services on a statically provisioned dedicated cluster is a different than creating services in mixed pools of dedicated and public resources. None of the above works considered this combination of resource pools.

7 Conclusion and Future Works

In this paper, we have presented the novel architecture of Public Computing Utility that is built by augmenting dedicated resource clusters with opportunistically available public resources. After characterizing the public resources and defi ning the resource management problem of this PCU architecture, we discussed the trade-offs among different strategies for managing the resources. Based on that we have devised a scheduling algorithm that allocates resources from both dedicated and public resource pools in order to maximize the goodput of the whole utility as well as the compliance with the user SLAs. We analyzed the system model from the perspective of queueing and reliability theories and derived relationship between different design parameters and system performance. We carried out extensive simulations to establish several design decisions for the architecture and the algorithm. Finally, for verifying the performance of the scheduling algorithm on the proposed architecture, we tested it with the compute intensive application workload from a university research Grid [2] and compared the resource utilization and job runtime elongation with the actual trace.

The results indicate that the PCU approach dramatically increases the virtual capacity of a computing utility service provider and enables the provider to substantially overbook for client service agreements. For example, we have seen that with very conservative provisioning of dedicated resources, we have more than 250% increase in average resource utilization compared to the dedicated resource based research Grid, with only minimal increase in execution time of the jobs on average. From the results we have also observed that the transmission load on the network that connects the dedicated resource pool to the Internet constrains the total size the dedicated resource pool. Therefore, to get the maximum benefi t from a given number of dedicated resources, we either need to provide enough network bandwidth, or distribute the resources in different networks. Another major factor that influences the performance is the actual capacity distribution and availability pattern of the public resources. A complete characterization of behavior of the public resources would help better tuning of the parameters for performance.

We have seen that preemptive migration is necessary for assuring service qualities for the clients. wrapping application programs in virtual machines may be a option for facilitating migrations, but a more lightweight wrapper that can monitor the application's resource usage as well as implement the migration protocol would be a better option for implementation for PCU.

References

[1] S. Asaduzzaman and M. Maheswaran, "Utilizing Unreliable Public Resources for Higher Profi t and Better SLA Compliance in Computing Utilities," *Journal of Parallel and Distributed Computing*, vol. 66, no. 6, pp. 796–806, 2006.

[2] "The Distributed Advanced School of Computing and Imaging (ASCI) Supercomputer 2 (DAS-2)," http://www.cs.vu.nl/das2/, Vrije Universiteit, Amsterdam.

[3] D. Jackson, Q. Snell, and M. Clement, "Core Algorithms of the Maui Scheduler," *Lecture Notes in Computer Science*, vol. 2221, p. 87, Jan. 2001.

[4] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International J. Supercomputer Applications*, vol. 15, no. 3, 2001.

[5] D. P. Anderson, "BOINC: A System for Public-Resource Computing and Storage," in *5th IEEE/ACM International Workshop on Grid Computing*, Nov. 2004.

[6] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[7] M. Ripeanu, "Peer-to-Peer Architecture Case Study: Gnutella Network," in *1st International Conference on Peer-to-Peer Computing (P2P01)*, Aug. 2001.

[8] S. A. Baset and H. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol," Department of Computer Science, Columbia University, Tech. Rep. CUCS-039-04, Sep. 2004.

[9] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: an Experiment in Public-resource Computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[10] D. P. Anderson and G. Fedak, "The Computational and Storage Potential of Volunteer Computing," in *6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, May 2006.

[11] M. W. Mutka and M. Livny, "The Available Capacity of a Publicly Owned Workstation Environment," *Performance Evaluation*, vol. 12, pp. 269–284, 1991.

[12] B. Chun and A. Vahdat, "Workload and Failure Characterization on a Large-Scale Federated Testbed," Intel Research Berkeley, Tech. Rep. IRB-TR-03-040, Nov. 2003.

[13] J. Brevik, D. Nurmi, and R. Wolski, "Automatic Methods for Predicting Machine Availability in Desktop Grid and Peer-to-peer Systems," in *IEEE International Symposium on Cluster Computing and the Grid (CCGrid'04)*, Apr. 2004.

[14] R. Wolski, D. Nurmi, J. Brevik, H. Casanova, and A. Chien, "Models and Modeling Infrastructures for Global Computational Platforms," in *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, Apr. 2005.

[15] L. Kleinrock and W. Korfhage, "Collecting Unused Processing Capacity: an Analysis of Transient Distributed Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 5, pp. 535–546, 1993.

[16] K. Czajkowski, I. T. Foster, and C. Kesselman, "Resource Co-Allocation in Computational Grids," in *High Performance Distributed Computing (HPDC)*, Aug. 1999.

[17] M. Maheswaran, B. Maniymaran, S. Asaduzzaman, and A. Mitra, "Towards a Quality of Service Aware Public Computing Utility," in *3rd IEEE Symposium on Network Computing and Applications: Adaptive Grid Computing Workshop*, Aug. 2004, pp. 376–379.

[18] A. Mitra, R. Udupa, and M. Maheswaran, "A Secured Hierarchical Trust Management Framework for Public Computing Utilities," in *15th Annual International Conference in the IBM Centers for Advanced Studies (CASCON)*, Oct. 2005.

[19] V. Loa, D. Zhou, D. Zappala, Y. Liu, and S. Zhao, "Cluster Computingon the Fly: P2P Scheduling of Idle Cycles in the Internet," in *3rd International Workshop on Peer-to-Peer Systems (IPTPS'04)*, Feb. 2004.

[20] N. Andrade, L. Costa, G. Germoglio, and W. Cirne, "Peer-to-peer Grid Computing with the OurGrid Community," in *23rd Brazilian Symposium on Computer Networks - IV Special Tools Session*, May 2005.

[21] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," in *2nd Symposium on Networked Systems Design and Implementation (NSDI'05)*, May 2005.

[22] J. Meehean and M. Livny, "A Service Migration Case Study: Migrating the Condor Schedd," in *Midwest Instruction and Computing Symposium*, Apr. 2005.

[23] J. G. Hansen and E. Jul, "Self-migration of Operating Systems," in *2004 ACM SIGOPS European Workshop*, Sep. 2004, pp. 241–299.

[24] D. S. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, "Process Migration," *ACM Computing Surveys*, vol. 13, no. 3, pp. 241–299, Sep. 2000.

[25] F. Leung, G. Neiger, D. Rodgers, A. Santoni, and R. Uhlig, "Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization," *Intel Technology Journal*, vol. 10, no. 3, 2006.

[26] M. Larabel, "Linux KVM Virtualization Performance," http://www.phoronix.com, Jan. 2007.

[27] D. A. Lifka, "The ANL/IBM SP Scheduling System," in *Workshop on Job Scheduling Strategies for Parallel Processing(IPPS '95)*. Springer-Verlag, 1995, pp. 295–303.

[28] A. Erlang, "Solution of some Problems in the Theory of Probabilities of Significance in Automatic Telephone Exchanges," *The Post Office Electrical Engineer's Journal*, vol. 10, pp. 189–197, 1918.

[29] S. M. Ross, *Introduction to Probability Models*, 5th ed. Academic Press Inc., 1993.

[30] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, B. Park, and H. Song, "Parsec: A Parallel Simulation Environment for Complex Systems," *Computer*, vol. 31, no. 10, pp. 77–85, Oct. 1998.

[31] H. Li, D. Groep, and L. Walters, "Workload Characteristics of a Multi-cluster Supercomputer," *Lectur Notes in Computer Science: Job Scheduling Strategies for Parallel Processing(JSSPP'04)*, vol. 3277, 2005.

[32] U. Lublin and D. G. Feitelson, "The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs," School of Computer Science and Engineering, The Hebrew University of Jerusalem, Tech. Rep. 2001-12, Oct. 2001.

[33] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys, *Handbooks in Operations Research and Management Science*. Elsevier Science Publishers, 1993, vol. 4, ch. 9, pp. 445–522.

[34] P. Bartley, M. Florian, and P. Robillard, "Scheduling with Earliest Start and Due Date Constraints," *Naval Research Logistics Qaurterly*, vol. 18, pp. 511–519, Dec. 1971.

[35] M. Garey and D. Johnson, "Two-processor Scheduling with Start-times and Deadlines," *SIAM Journal on Computing*, vol. 6, pp. 416–426, 1977.

[36] ——, *Computers and Intractability: A Guide to the theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[37] M. L. Dertouzos and A. K. Mok, "Multiprocessor On-line Scheduling of Hard-Real-Time Tasks," *IEEE Transactions on Software Engineering*, vol. 15, no. 12, pp. 1497–1506, 1989.

[38] J. Sgall, “On-Line Scheduling – a Survey,” in *Online Algorithms – The State of the Art*. Springer Verlag, 1997, pp. 196–231.

[39] D. P. Anderson, E. Korpela, and R. Walton, “High-Performance Task Distribution for Volunteer Computing,” in *First IEEE International Conference on e-Science and Grid Technologies*, Dec. 2005.

[40] R. Raman, M. Livny, and M. Solomon, “Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching,” in *12th IEEE International Symposium on High-Performance Distributed Computing*, Jun. 2003.

[41] D. Zhou and V. Lo, “Wave Scheduling: Scheduling for Faster Turnaround Time in Peer-to-peer Cycle Sharing Systems,” in *Workshops on Job Scheduling Strategies for Parallel Processing (JSSPP’05)*, Jun. 2005.

[42] C. Kenyon and G. Cheliotis, “Creating Services with Hard Guarantees from Cycle Harvesting Resources,” in *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID’03)*, May 2003.

[43] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger, “Oceano – SLA Based Management of a Computing Utility,” in *7th IFIP/IEEE International Symposium on Integrated Network Management*, May 2001.

[44] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke, “SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems,” *Lecture Notes in Computer Science*, vol. 2537, pp. 153–183, 2002.

[45] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, “Managing Energy and Server Resources in Hosting Centers,” in *18th ACM Symposium on Operating Systems Principles*, Oct. 2001.

[46] S. Ranjan, J. Rolia, and E. Knightly, “QoS Driven Server Migraion for Internet Data Centers,” in *IWQoS 2002*, May 2002.

[47] M. Aron, P. Druschel, and W. Zwaenepoel, “Cluster Reserves: A Mechanism for Resource Management in Cluster-Based Network Servers,” in *ACM SIGMETRICS*, Jun. 2000.