Available online at www.sciencedirect.com

SCIENCE DIRECT®

J. Parallel Distrib. Comput. 66 (2006) 796–806

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

Utilizing unreliable public resources for higher profit and better SLA compliance in computing utilities

Shah Asaduzzaman, Muthucumaru Maheswaran*

School of Computer Science, McGill University, Montreal QC, Canada H3A 2A7

Received 18 October 2004; received in revised form 11 June 2005; accepted 27 January 2006
Available online 20 March 2006

Abstract

Computing utilities are emerging as an important part of the infrastructure for outsourcing computer services. Fundamental to outsourcing is the notion of quality of service, which is defined by service level agreements (SLAs) between the computing utilities and clients. One of the major objectives of computing utilities is to maximize their net profit while maintaining customer loyalty. To achieve this objective, the computing utilities should meet or exceed their SLA constraints most of the time. Defining the SLAs conservatively might be one way of easily achieving these goals. However, by tuning the SLA parameters conservatively the computing utility might under utilize its resources with a resultant loss of revenue. Therefore, we can see two main issues with SLA management: designing SLAs competitively so that expected revenue for the computing utility is maximized and maintaining the operating conditions such that SLAs are satisfied with very high probability. In this paper, we show that inducting unreliable public resources into a computing utility enables more competitive SLAs while maintaining higher levels of run time compliances as well as maximizing profit. Our scheduling algorithms assume that idle cycles from public resources are available in plenty, therefore, the performance gains do not incur any additional financial cost. However, there is communication overhead when public resources from a wide area network is included. This overhead is kept to the minimum by enabling the scheduler work without any monitoring on the public resources.

Keywords: Resource management; Scheduling; Service level agreement; Public computing utility

1. Introduction

Constant improvements in computer communications and microprocessor technologies are driving the development of new classes of network computing systems. One such system is the *computing utility* (CU) that brings large number of resources and services together in a virtual system to serve its clients. Typically, CUs are built by connecting the resources or services to a *resource management system* (RMS) that itself is implemented either centrally or federally. The RMS allocates resources to the client requests such that some measure of delivered performance is maximized subject to fairness constraints. The CUs have diverse designs based on various parameters including target applications, organization of the RMS, classes of resources managed by the utility, and levels of services offered to the clients.

Although CUs were initiated in the realm of high-performance computing [13], they have been successfully adopted in other areas such as online gaming, content distribution, and video-on-demand. While research into CUs is striving to achieve application independent designs [7], target applications continue to cast strong influence on the design of CU systems. The organization of the RMS is another key CU design consideration. The organization of the RMS can impact the scalability, extensibility, and fault tolerance of the CU. The CU can manage different classes of resources. For instance, a CU can include dedicated resources that are owned and exclusively managed by the CU, volunteer resources that are not bound by any contracts, and partially committed resources that are managed through incentives schemes administered by the CU. The levels of services offered to the clients by the CU is another important consideration. In the simplest case, the CU offers *best effort* services to the clients. However, to attract

* Corresponding author. Fax: +51 43983883.
E-mail addresses: asad@cs.mcgill.ca (S. Asaduzzaman), maheswar@cs.mcgill.ca (M. Maheswaran).
URL: http://www.cs.mcgill.ca/∼anrl/ (M. Maheswaran).

clients with business critical applications the CUs should offer services with *quality of service* (QoS) assurances.

One of the common approaches to construct CUs is to deploy server resources and run CU infrastructure software such as Grid [15] and Web Services [6] on them to create virtual systems for research [23] and commercial purposes [11]. When server resources are deployed by a single organization for the sole purposes of the CU, proper capacity planning is essential to make CU cost-effective while meeting the performance expectations. When multiple organizations are contributing resources towards the CU, the management becomes complicated unless each organization sets aside certain number of resources exclusively for the CU. PlanetLab [23] is one example of CU that falls into the latter category.

This paper is concerned about augmenting CUs using "public" resources (i.e., resources that wish to contribute their computing, storage, and network capacities without subjecting themselves to any contractual agreements). Several large-scale network computing applications including *peer-to-peer* (P2P) file sharing systems such as Gnutella [21] and volunteer computing systems such as SETI@home [1] have demonstrated the tremendous potential of using public resources. However, these systems also illustrate the challenges that need to be addressed in using public resources. The additional capacity introduced by the public resources should be efficiently managed to provide higher levels of services in a cost-effective manner. In this paper, we refer to the augmented CU where public resources provide additional capacity as the *public computing utility* (PCU).

The PCU model we use here assumes a finite-sized private (dedicated) resource pool and a very large (nearly infinite) sized public resource pool. The private resource pool may be either distributed or concentrated at a single network location. The public resource pool, on the other hand, is fully distributed and is organized in a P2P network. A P2P discovery service [27] is assumed to be available to locate the most appropriate set of public resources to satisfy a given request. Although public resources are available in plenty, their service rates are unpredictable. Estimates of expected performances based on observation of prior engagements provide the only basis for choosing the best candidates from the available public resources.

The clients using a CU subscribe through a CU service provider with a *service level agreement* (SLA) that defines among other parameters the maximum load the client is permitted to offer and the portion of the offered load guaranteed to be delivered by the provider. The SLA also defines the pricing scheme for the service and penalties that should incur in case of a quality violation by the CU. The parameters of the SLA guide the RMS in deciding resource allocations.

Although different applications can potentially use a PCU, here we consider only high-throughput computing applications. In this situation, job requests belonging to different clients arrive at the PCU at arbitrary times. Each job has a deadline before which it should be completely serviced if the client is to receive full benefit. It is well known [20] that even if all the information about the jobs (i.e., arrival time, processing time, and deadline) are known a priori, finding the optimal non-preemptive schedule that maximizes throughput is a NP-hard problem for a multiprocessor system. In a practical PCU setting, the RMS has to take the allocation decision as soon as the jobs arrive, and the arrival times are arbitrary.

In this paper, we devise an online scheduling heuristic for the RMS of the PCU. The PCU online heuristic needs to decide what class of resources (public or private) should be used for servicing a given request in addition to determining how best to use the selected resource. Because the PCU is bound by the SLAs when delivering services to the clients, we need to consider the SLAs in the resource allocation process as well.

Section 2 discusses the related results found in the literature. Section 3 explains the proposed system architecture in detail. Section 4 defines the resource scheduling problem being dealt with in the PCU. Section 7 discusses the results from the simulations performed to evaluate the resource allocation alternatives. Section 6 describes a proposed PCU architecture and demonstrates the use of the heuristic for QoS in the PCU system context.

2. Related work

Multiprocessor job scheduling is a well-studied problem in operations research and computer science. Although several optimal algorithms are available [20] 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 [5]. Also for the two processor case arbitrary jobs with certain precedence constraints can be scheduled in polynomial time [17]. However, scheduling jobs with arrival time and deadline constraints is proven to be a NP-hard problem for more than two processors [18]. In fact [10] 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 [28].

One major goal of the RMS of a PCU is to enforce QoS according to the SLAs signed up with its clients. SLA compliant resource management for cluster of dedicated machines has been studied in several research projects. Oceano [3] has described the architecture and protocols for SLA management in cluster based hosting utilities, whereas [9] gives a SLA negotiation protocol for more distributed computing system such as the Grid [14,15]. 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 [4,8,16,26] with different optimization goals in different cases. For instance, [8,26] attempt to minimize the total number of active servers to minimize energy consumption and to maximize average server utilization, respectively, whereas [4] targets performance isolation. An SLA driven resource management scheme for clusters to maximize net profit is evaluated in [16]. Further, [29] defines QoS through aggregate yield functions and evaluates a greedy heuristic to manage resources in a server cluster to maximize yield as well as reserving a minimum resource share for each service classes. But none of the above system has dealt with QoS on top of unreliable resources like a PCU.

The Condor project [30] focuses on harvesting unused resources from heterogeneous public machines, but their resource management mainly emphasizes on discovery and co-allocation of resources through matchmaking [24] and gangmatching [25]. They do not support SLA driven QoS aware resource management on the public resource pool.

One work that is very close to our work is [19], which examines the resource management problem of providing marketable stochastic QoS using an infinite pool of public resources of stochastic behavior and a finite pool of dedicated private resources. Although their architecture matches closely to that of PCU, our work is significantly different from theirs in several dimensions. In their model the public resources are homogeneous in performance but cycle lengths are stochastic (i.e., the public resources deliver at a fixed and known performance level when they are connected to the system). Instead we have modeled the throughput of the public resources to be stochastic (i.e., the performance levels delivered by the public resources vary even while they remain connected with the PCU), which more closely captures the real behavior. The QoS guarantee provided by their scheme is a stochastic QoS measured in quantiles [19], where the probability distribution of the delivered QoS (cycle lengths) is identical to the cycle length distributions of the individual public resources. Further, their scheme does not have any long-term SLA with the clients. As performance evaluation they have studied the expected amount dedicated resources required for completing the jobs within the deadline constraints.

In presence of SLAs with clients as is the case in our PCU model, the SLA compliance is a major issue because non-compliance incurs penalties for the provider. Our scheduling heuristic is devised to simultaneously maximize the net-profit of the service provider and the level of compliance with the long-term SLAs. Our work also differs from [19] in the jobs characteristics. Instead of correlating the cycle-length requirements of the jobs and the public resource characteristics, we deal with job characteristics that are independent of the underlying public resource characteristics. Our investigation also includes job streams arriving from multiple clients that have different SLAs with the PCU.

3. The PCU system model and assumptions

The primary component of the PCU is a proximity aware P2P network such as the Pastry [27] that connects all the resources that participate in the system. The P2P network enables efficient resource discovery by finding the most appropriate sets of resources in response to queries posed by resource brokers working on behalf of clients. The public resources are expected to be uniformly dispersed throughout the Internet and the private resources can be concentrated as clusters at certain locations on the Internet. There can be one or more such clusters that are installed and maintained by the PCU provider. For uniformity and simplicity, the public and private resources are both discovered through the common P2P discovery network. The resource broker that is seeking the resources on behalf of the client decides in conjunction with the PCU which type of resource should be used to service a given request from the client.

Several issues should be addressed in developing the resource allocation process in a PCU system. These include: (a) co-allocating different resources such as processing bandwidth, storage capacity, and network bandwidth, (b) using trust measures of public resources to derive robust resource allocations, (c) 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 congestions on the QoS delivered by the PCU, 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 only one resource, the CPU processing bandwidth. We model the public resources to have stochastic throughput with identical probability distribution. The parameters of the distribution are estimated from the history information on the performance of the public resources. It is known that proper incentive management will modify the default behavior of the volunteer resource providers to a more favorable one for the PCU [2,12]. This influence of incentive management is not considered in this paper.

Job requests from different clients arrive at the PCU at different points in time. The RMS of the PCU makes allocation decisions at the end of discrete time quantums called *epochs*. The length of an epoch is a design parameter for the system. The allocation decisions of the RMS are influenced by several parameters including: (a) current utilization of the private resource pool administered by the PCU, (b) current load offered by the different clients, (c) current value of the expected performance of the best-effort resources, and (d) throughput guaranteed to the particular client by the PCU in its SLA.

Another PCU design consideration is performance monitoring of the public resources. In the current PCU RMS design, we assume that no progress monitoring facility exists at a public resource to determine the rate at which the public resource is working for the PCU. The only feedback we obtain from the public resource is when it completes a job or job component. If we can have progress indicators from the public resources, we can improve the contribution from the public resources towards overall throughput.

4. The resource management problem

Computational jobs arrive from each client of the PCU service provider at arbitrary points in time with each job consisting of arbitrary number of mutually independent (i.e., parallel) components. Along with the jobs, clients are assumed to submit an estimation of the workload of each of the components at the submission time. Components are possibly of different sizes. An overall deadline is defined for the job before which all the components must finish their execution.

The SLA that is signed off-line between the provider and a client reserves a throughput guarantee for the corresponding client. The SLA defines various parameters including:

- ρ, the ratio of the client-offered workload that is guaranteed to be carried out by the PCU service provider.
- V, the maximum limit on the workload that can be offered by the client.

From these parameters it can be deduced that when the offered load is not more than V, the delivered throughput should be ρv or more in order to be compliant with the SLA. If offered load v is greater than V, it is sufficient for the PCU to deliver ρV amount of throughput.

Another service objective of the PCU is meeting the deadlines of the individual jobs presented by the clients. The PCU provider earns revenue in proportion to the total delivered computational work for the jobs that finish completely within their deadline (with all of its components). A global throughput versus price for unit work curve defines this revenue. The curve may be concave to emphasize the fact that the price is higher for work in higher throughput, but the rate of increase is gradually slowed down. Further, there is penalty for violation of the SLA terms and the penalty is proportional to amount of deviation of the delivered throughput from guaranteed throughput, measured over a specified time window. The length of the window and a moving averaging factor α_{SLA} that is used to smooth out the burstiness in offered and completed workloads across the windows are defined as SLA parameters.

Guided by the above service objectives, the job scheduling component, which is the core component of the RMS, has precisely two distinct responsibilities at the end of each epoch:

- Accept or reject the jobs that arrived during the last epoch, and start the components of the accepted jobs on the private and/or public resources.
- Migrate some components of some jobs that are vulnerable for deadline violation, from public resources to the dedicated resources. Because there is no checkpointing and no progress monitoring of the jobs running on public machines, the RMS has to *restart* the job from the beginning instead of relocating the remaining portions.

The optimization goal of the job scheduler is to maximize the net revenue (i.e., revenue–penalty) of the PCU service provider. To achieve this objective, the job scheduler has to maximize the amount of work done for jobs that do not violate the deadline. Another goal is to ensure the fairness among the clients, so that all of them have equal treatment from the scheduler in accordance with the subscriptions agreed upon in the SLAs. The next section describes the heuristic solutions we devised to achieve these goals.

5. Heuristic solutions for resource management

In this section, we present three heuristic solutions to resource management in a PCU environment. The first solution, the PCU heuristic, is proposed as part of this work. The next two solutions are adopted from the scheduling literature for the PCU environment for comparison purposes.

5.1. PCU heuristic: an online resource allocator

The scheduler of the RMS uses an online heuristic to take decisions about allocating available resources to incoming jobs. To reduce the scheduling overhead, the RMS executes the scheduling rules at discrete points of time (i.e., at the end of each scheduling epoch δ). Another component of the RMS, the SLA monitor measures the current deviation D_c of delivered throughput from required throughput for each client c, according to the SLA specified time-window τ_c and moving average factor α_{SLA}. Say the total arrived workload in a time-window is W_{a} and total completed and delivered workload is W_{d}, both W_{a} and W_{d} being smoothed by moving average with the past values. Then,

$$D_c = \max(V_c, W_{\text{a}}\rho_c) - W_{\text{d}}.$$

In the above equation, V_c and ρ_c are SLA-defined maximum load and acceptance ratio for client c. The current value of D_c is available to the scheduler at the end of every epoch. There are two parts of the decision taken by the scheduler at the end of every epoch: (i) accept newly arrived jobs and start them on public and/or private resources, and (ii) relocate and restart the deadline vulnerable jobs from public resource to the private resource pool (in absence of checkpointing and progress monitoring, it is impossible to migrate without restarting).

5.1.1. Acceptance of jobs

For each client, the scheduler maintains a priority queue for newly arrived jobs, ordered by highest contributing job first. For a job with total workload W and total available time T_{a} before deadline, the throughput contribution is $\frac{W}{T_{\text{a}}}$. Every time the foremost job from the queue of the client having highest $D_c - W_c$ value is chosen, where W_c is the amount of workload so far accepted for client c in current SLA window.

All the jobs are ultimately accepted, and each of them is assigned one of the two different levels of *restart-priority*, which is used latter for restarting decisions. The jobs are accepted according to the following rules:

(1) As long as available dedicated resources allow, schedule jobs with *critical* components on dedicated and the rest on public resources. The components that are expected to violate deadline if scheduled on a public resources according to their currently estimated expected throughput μ, are identified as critical components. Among the M private resources, M_{r} are reserved for restarting phase (the ratio $\frac{M_{\text{r}}}{M}$ is a design parameter). Let M_{o} denotes the number of occupied private resources at any given time and m denotes the number of critical components in the new job. Accepting jobs with this rule continues as long as $M_{\text{o}} + m \leqslant M - M_{\text{r}}$, where m . Otherwise, the scheduler switches to the rule-2. The restart-priority is set to high for all the jobs scheduled by rule-1.

(2) For the rest of the enqueued jobs all components are scheduled on public resources. For any client c, as long as total accepted workload from that client in the current SLA window is below $\rho_c V_c$, the restart-priority of the accepted job is high, otherwise it is low.

5.1.2. *Restart jobs*

At the end of every epoch, the scheduler restarts some deadline-vulnerable job-components from public resources. At any given time, a job component is defined to be deadline-vulnerable if it cannot be completed before deadline unless it is allocated a dedicated resource right at that time. A priority queue is maintained for all the vulnerable components. The queue is ordered descending primarily by the restart-priority (explained earlier) and secondly by violation probability ($p_{\rm v}$). $p_{\rm v}$ is computed at the job-launch time from the available information (distribution of the public resource throughput, component size and the deadline). From the queue, high restart-priority components are restarted as long as any dedicated resource is available. Low restart-priority components are restarted as long as available dedicated resource is greater than $M_{\rm r}$. The rest of components are left on public resource.

5.2. *Least laxity first and greedy heuristics*

For performance evaluation we compare our PCU heuristic with the well known *least laxity first* (LLF) [28] heuristic and a greedy heuristic. We use the LLF heuristic to schedule the jobs only in the private pool of resources. The laxity is the slack between possible execution finish time and deadline. New jobs from each client enter a separate priority queue where the priority is laxity of job's deadline. At every epoch jobs are popped from the queues and scheduled in dedicated resources if available. Otherwise, the job is deferred until the time after which it becomes infeasible to execute within deadline. As a fairness scheme the queue of the client with highest deviation from SLA is favored when choosing every job.

The greedy heuristic, another one that we used for comparison, works on the same PCU architecture with a combination of private and public resource pools. The greedy scheduling policy chooses jobs from the arrival queues in every scheduling epoch in the order of highest contributing job of the highest deviating client first. It schedules all components of incoming jobs on private resources in the order of longer component first, as long as there is spare capacity in the private resource pool. All the remaining job-components are scheduled on public resources until all the arrival queues are exhausted.

6. Architecture to deploy the heuristics

It is useful to outline a system architecture where the heuristic may be deployed to extract the desired gain from public resources. In this section, we describe an architecture for the PCU and show how quality of service support can be implemented in the resource management layer. This research is part of a bigger project which is targeted to build a complete PCU named Galaxy.

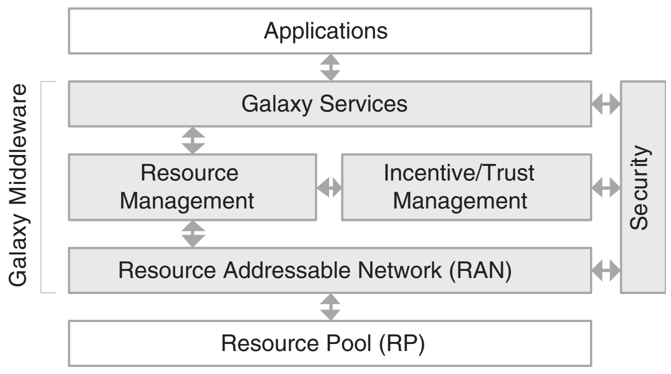

Fig. 1. The Galaxy architecture.

6.1. *A public computing utility architecture*

The proposed architecture for the Galaxy PCU is shown in Fig. 1. The lowest layer of the architecture is the P2P overlay network called the *resource addressable network* (RAN). All the resources that participate in the PCU plug into the RAN. The RAN provides the resource naming, discovery, and access services to the PCU. The next upper layer is the *Galaxy resource management system* (GRMS). Similar to the RAN, the GRMS is also organized as a P2P overlay network of managerial entities called *resource brokers* (RBs). In the RAN, the peers are virtualized resources whereas in the GRMS the peers are RBs. The trust/incentive management is a collaborating module to the GRMS. It controls the behavior of resources, especially the public resources, in the system. The next upper layer is the Galaxy services. Although the architecture does not impose any restriction on the organization of this layer, it could be organized as a P2P network. Example Galaxy services include application level QoS managers, shell interfaces, and network file systems. Security is a layer in this Galaxy middleware that spans all the other layers in parallel to provide the system from malicious activities (external and internal to the system).

6.2. *The resource management layer*

The GRMS layer, which is responsible for the resource management in the Galaxy PCU, is composed of two sets of entities: (a) *resource peers* (RPs) and (b) RBs. RPs are GRMS layer representatives of the resources present in the Galaxy. The resources RPs represent can be individual resources, cluster of resources, virtual resources, or software entities. RPs are the mediators in the GRMS resource allocation: they launch resource requests on behalf of the entities they represent and regulate the resource acquisition requests received for the resources under their control. Aggregating multiple resources under an RP has number of architectural benefits: it matches the administrative domains present in real world and provides an easy way of handling trust levels and incentives. The role of RBs and RPs are illustrated in Fig. 2.

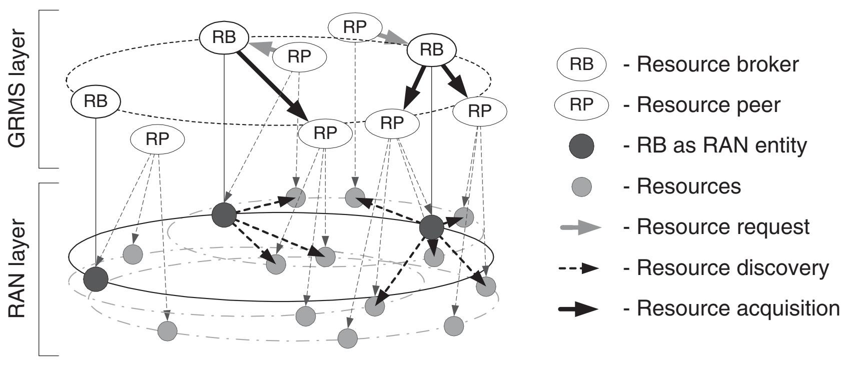

Fig. 2. GRMS layer functionalities.

The RBs are the entities that coordinate most of the activities within the GRMS layer. Any RP can act also as an RB as long as it has sufficient reputation such that existing RBs accept the new one. RBs have their virtual representations in the RAN level and this enables the RBs to use the RAN's scalable resource discovery mechanism to discover resources and other RBs. Two major functionalities of the RBs are to search and allocate resources as requests emerge from RPs and to assign and revalidate trust levels and incentive shares of the RPs. Each RP chooses an RB (probably the closest) to send the resource requests. The RB uses the RAN discovery mechanism to discover the appropriate resources. The discovered resources tell the RB who their RP(s) are. The RB then mediates with the destination RPs to acquire the resources for the requesting RP. At the end of the resource utilization, the donor and client RPs report the performance during the utilization back to the RB and based on which the RB readjusts the incentive shares owned by the donor RPs.

6.3. Implementing QoS support in resource management: a scenario

The GRMS layer implements the "core" set of functionalities to manage the QoS delivered by Galaxy. The big part of the Galaxy resource pool made up of publicly owned resources that are guided by incentives and reputation. But, since it is hard to guarantee QoS with uncontrolled public resources alone, Galaxy engineers the QoS guarantees by augmenting the public resources with a fully controlled and reliable pool of dedicated resources.

We implement this idea in conjunction with the QoS guaranteed resource scheduling service provided by the RBs. Followings are the steps of a guaranteed resource allocation scenario.

- The RP who needs resources to launch their jobs contacts a nearby RB for necessary resource allocation.
- The RBs are the resource allocator in Galaxy and to meet the clients' request it may assign some resource from the large set of public resources that are discovered through RAN. Alternately, RB may reserve some of the trusted resources from the RAN, and allocate from this reserved pool in urgent situations.
- In case of failure or poor performance of a public resource, RB may restart the job on one of the trusted resource from the reserved pool. The heuristic described in Section 5.1 may be used here to decide which job needs to switch resource urgently.

Thus RBs try to maximize the job throughput as well as maximally fulfill the implicit service level agreement with the RPs that is defined with the resource requests.

7. Simulation results

We have evaluated the performance of the PCU heuristic through a simulator written in Parsec [22] by changing different parameters and comparing it with the greedy and LLF heuristics. In our simulation setup, the service provider had a pool of 100 dedicated machines and an infinite pool of public machines. There were five independent clients each feeding a stream of parallel jobs that should be completed within the given deadlines and having its own SLA. Job arrival is a Poisson process, with each job having a random number (k) of parallel components (geometrically distributed with a mean 25, unless mentioned otherwise). Each component of a job also has a random workload that is from a geometric distribution. Each job has a feasible deadline, i.e., it can always be completed if all the parallel components run on dedicated machines. Unless stated otherwise, the deadline was computed with a uniform random laxity between 0.5 and 2 times the mean component length, from the longest component. This tight deadline allows one trial on the public pool and failing that it should be restarted on a private resource.

All private machines have homogeneous throughput, completing 1 unit of workload of a component per second. The public resource throughput is sampled from Lognormal distribution with standard deviation 1.0 and mean less than 1.0, generally 0.8 unless stated otherwise. Justification behind using lognormal distribution is that being left skewed it closely resembles the behavior of the resources in a PCU setting, where most of the public resources may have very low or even 0 throughput.

In this section, we show the results of 2 sets of simulation experiments. The first five graphs (Figs. 3–7 in Section 7.1) shows the comparative study of the performance of our new

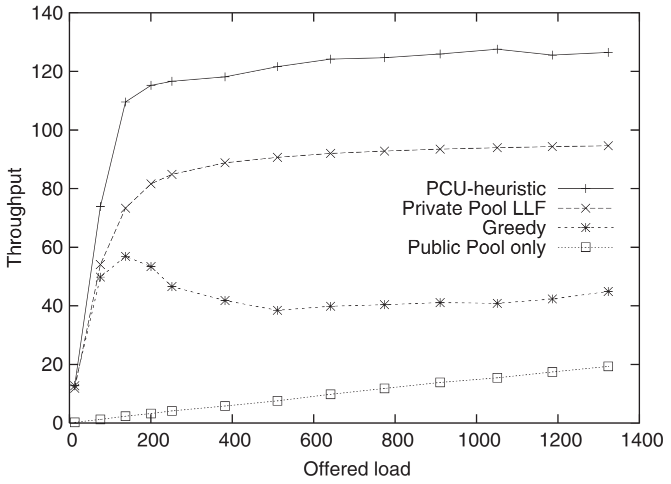

Fig. 3. Variation of mean throughput with offered load values for mean public resource throughput $\mu = 0.80$, mean number of parallel components $P = 25$, total number of private resources $M = 100$, and total SLA booking, $\sum \rho V = 100$.

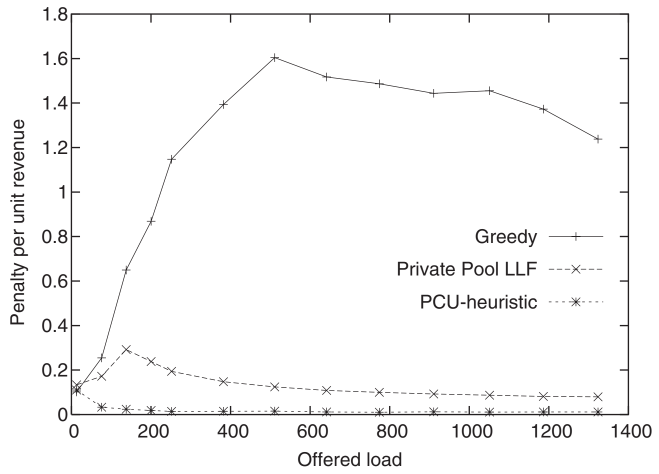

Fig. 4. Variation of penalty per unit revenue with offered load for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$.

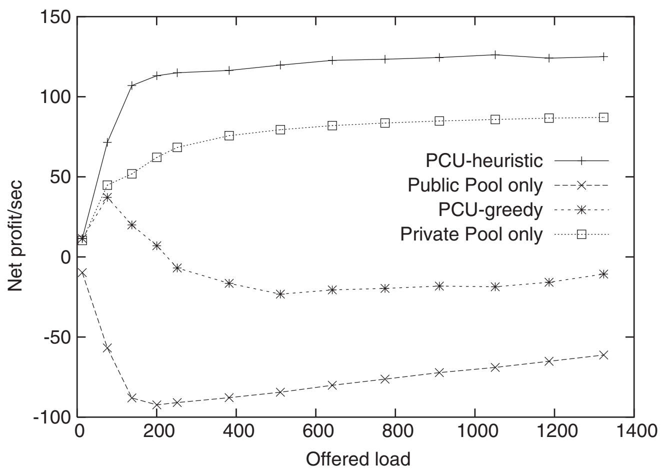

Fig. 5. Variation of net profit earned by the PCU provider with offered load for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$.

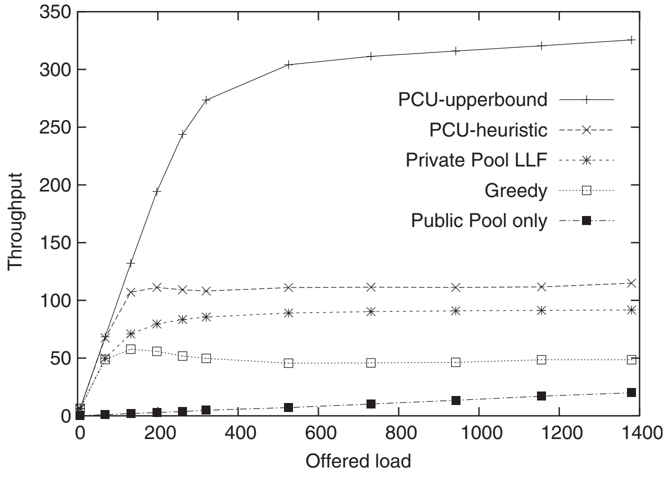

Fig. 6. Upper bound on PCU throughput assuming future behavior of public resources is known for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$.

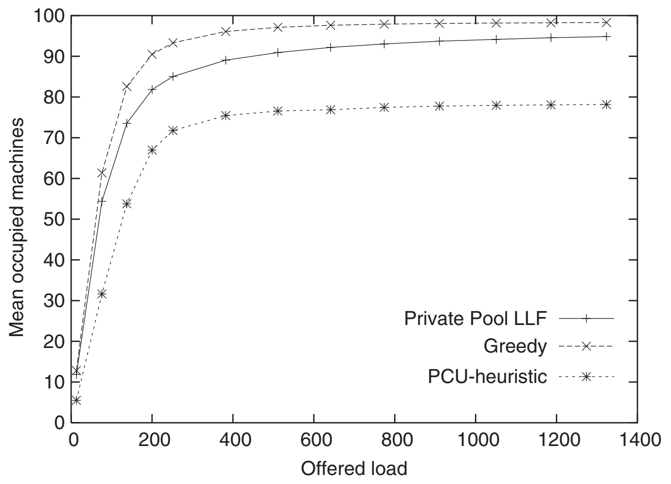

Fig. 7. Utilization of dedicated resources versus offered load for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$.

heuristic with other standard scheduling algorithms. The next nine graphs (Figs. 8–16 in Section 7.2) shows the study of how performance of the scheduler is affected by the change of different environment parameters.

7.1. Comparative study

In the first set of experiments the PCU heuristic is compared with LLF and greedy using throughput (Fig. 3), SLA compliance (measured using penalty per unit revenue in Fig. 4) and net revenue in Fig. 5. The PCU heuristic delivers better throughput than LLF, which implies it useful to augment public resource in a CU. Also the PCU-heuristic is superior in performance to the greedy heuristic in similar setting.

The penalty is higher with the LLF algorithm on private pool only system than the PCU heuristic, because jobs are not deprioritized when the client is offering more workload than the SLA upper bound. In case of the greedy algorithm, penalty

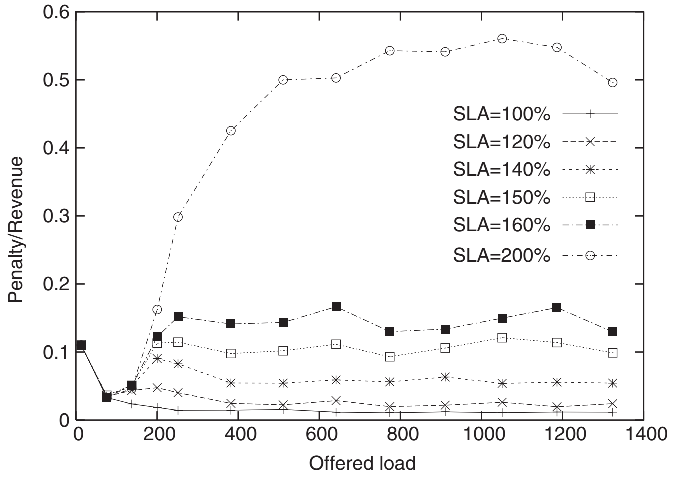

Fig. 8. Penalty per unit revenue earned at different levels of SLA booking for $\mu = 0.80$, $P = 25$, and $M = 100$.

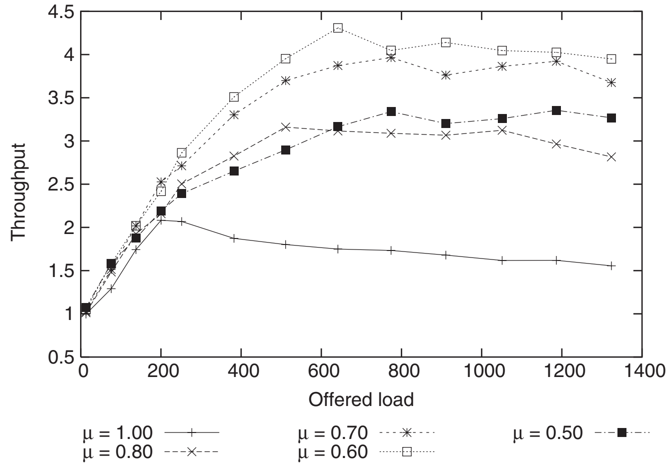

Fig. 11. Throughput gain at different public resource characteristics, with respect to the greedy resource allocation policy on combined pools for $P = 25$, $M = 100$, and $\sum \rho V = 100$.

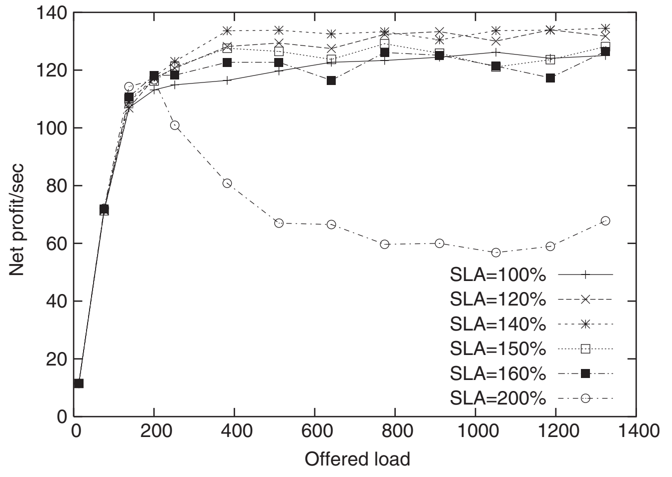

Fig. 9. Penalty per unit revenue earned at different levels of SLA booking for $\mu = 0.80$, $P = 25$, and $M = 100$.

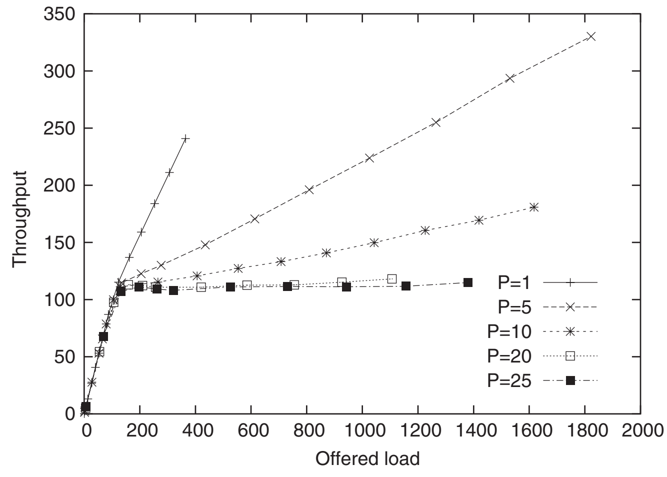

Fig. 12. Throughput gain at different public resource characteristics, with respect to the greedy resource allocation policy on combined pools for $P = 25$, $M = 100$, and $\sum \rho V = 100$.

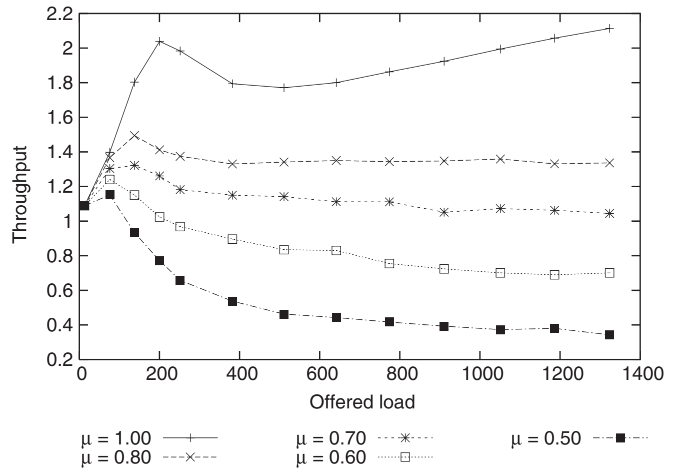

Fig. 10. Throughput gain at different public resource characteristics, with respect to a dedicated pool only system for $P = 25$, $M = 100$, and $\sum \rho V = 100$.

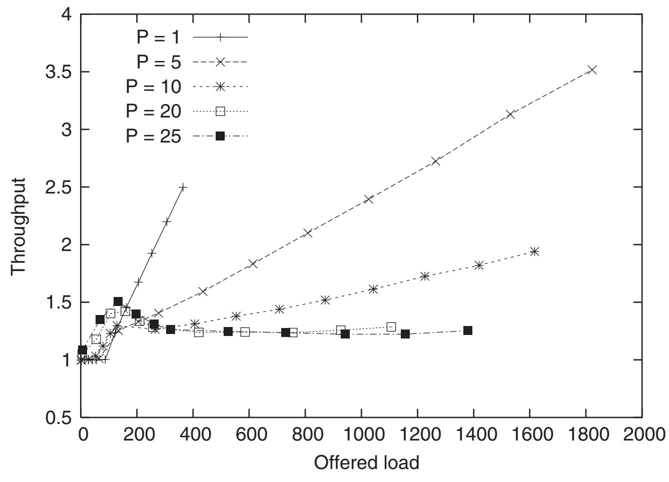

Fig. 13. Throughput gain at different degrees of parallelism, with respect to a dedicated pool only system for $\mu = 0.80$, $M = 100$, and $\sum \rho V = 100$.

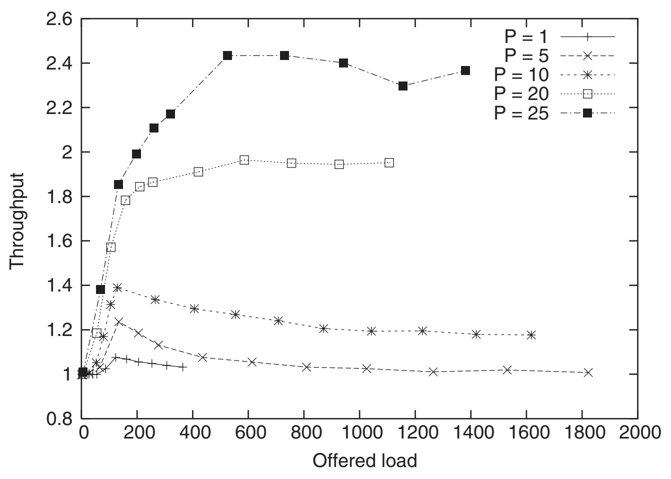

Fig. 14. Throughput gain at different degrees of parallelism, with respect to the greedy resource allocation policy on combined pools for $\mu = 0.80$, $M = 100$, and $\sum \rho V = 100$.

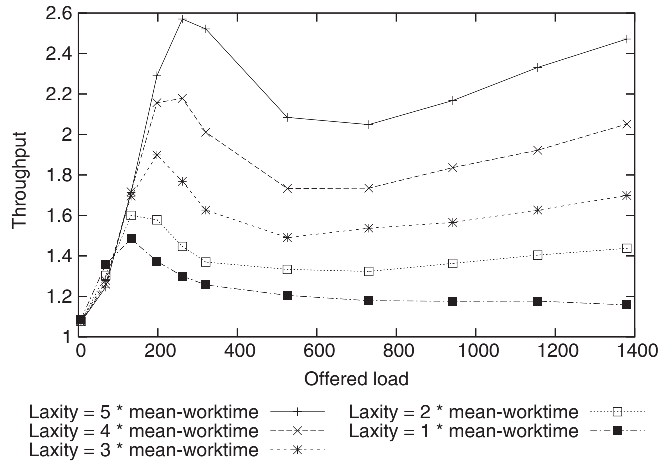

Fig. 15. Throughput gain at different amount of laxity in deadline, with respect to a dedicated pool only system for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$.

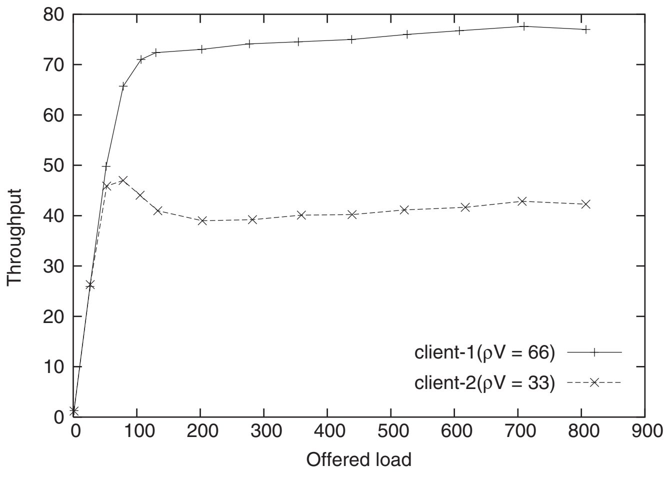

Fig. 16. Comparing delivered throughput to 2 clients having different max-load defined in SLA for $\mu = 0.80$, $P = 25$, and $M = 100$.

grows even higher when the client is overloading, because the dedicated pool gets fully occupied and most of the newly arriving jobs are put on public resources. Consequently, only a small portion of the newly arriving jobs can finish before their deadlines.

Fig. 6 shows that a much higher gain in throughput is achievable, if the exact knowledge of throughput of each public machine is available at schedule time, because then there is no need for restarting jobs. How far of this gain can be achieved without a priori knowledge of public resource characteristics, remains a problem for future research.

As Fig. 7 shows, the utilization of dedicated resources is higher for the greedy policy. This is because greedy uses the dedicated resources exhaustively. The PCU heuristic tries to execute a job-component primarily using public resources unless it becomes vulnerable for deadline violation. Also, in PCU, to allow the restarting of vulnerable components, it reserves a portion of the dedicated resources (25%) as contingency resources. These factors lower the utilization of dedicated resources in the PCU heuristic. Greedy's utilization is even more than LLF, because, in LLF jobs are not allocated unless the all the components fit in the private resources, whereas, greedy may put part of a job in private pool and rest in public pool.

7.2. Response to parameter changes

Here we consider the effects of different environment parameters like public resource capacity, SLA-overbooking, degree of parallellism of the jobs, etc. on the performance of the scheduler. First, to consider the flexibility in SLA overbooking, if the total agreed upon deliverable throughput (ρV) is higher than the maximum system capacity, the SLA deviation goes very high leading to correspondingly high penalties (Fig. 8). This in turn reduces the net profit earned by the service provider. From Fig. 9 it can be observed that SLA booking should be at 140% of the dedicated pool capacity to maximize the performance for the given PCU configuration.

Fig. 10 shows that use of PCU-heuristic brings gain in delivered throughput in most region of the spectrum of public resource behavior. It should be noted that with lognormal distribution, even if the mean throughput is equal to that of a dedicated machine, 62% of the public resources have throughput less than that of a dedicated machine. For very low public resource throughput, almost all of the jobs scheduled there need to restart, and since restart is subject to availability in the limited capacity private pool, many jobs get discarded. This explains the less than one throughput-gain with poor quality of public resources. Fig. 11 shows that PCU-heuristic outperforms the greedy heuristic across the whole spectrum.

Studying the effect of parallelism Fig. 12 shows that the effect is insignificant in underloaded situations, but when the system is overloaded, high number of parallel components increase the probability of failure of a whole job due to failure of only one or few components which could not be restarted when necessary. Hence, the total delivered throughput becomes low. Fig. 13 shows that the throughput with PCU heuristic always outperforms the LLF heuristic at a large degree for jobs with fewer parallel components. As we compare the PCU and the greedy heuristics in Fig. 14, it reveals that greedy heuristic performs much poorer with highly parallel jobs than the PCU heuristic. This is because in greedy, the private pool of resources gets occupied very quickly and a large number of jobs are scheduled on the unreliable public resources.

Study on the effect of laxity before deadline (Fig. 15) shows that throughput gain is much higher with relaxed laxity jobs. This is because with relaxed laxity the probability of getting a job component completed before deadline on a public resource increases, which incurs less restarts and better contribution from public resources.

Fig. 16 demonstrates the fairness of the PCU heuristic. For two different clients having SLA guaranteed max-workload (V) defined at 2:1 ratio but offering load at similar rate, it shows that the delivered throughput is proportional to the SLA-defined maxload of the clients, in overloaded situations. Thus the algorithm honors the SLA for the clients and distributes the available resources in a fair ratiometric way.

In summary, the results show that the PCU heuristic outperforms the other two standard algorithms in terms of throughput gain, SLA compliance and profit maximization. The current algorithm uses the public resources without any prior knowledge about their performance, which greatly reduces the communication overhead. It is shown that with the accurate a priori knowledge, much higher throughput is achievable. This tradeoff between communication overhead and performance gain is not studied in this paper. Resource utilization for the PCU heuristic is lower than other algorithms, because they use dedicated resources more exhaustively. On the other hand, this little under-utilization yields higher performance in terms of throughput. It is shown that the performance (throughput and SLA compliance) depends on different factors like the capacity of the public resources, the degree of parallellism of jobs and the amount of laxity allowed between job execution time and deadline. The algorithm also allocate the available resources fairly among competing clients, according to the commitments in SLA.

8. Conclusion

In this paper, we presented the idea of creating a PCU by augmenting computing utilities that are created using dedicated resources, with public resources. A resource management strategy for such an augmented system was presented. We proposed a resource allocation heuristic that uses the public and private (dedicated) pools of resources in an efficient manner. We carried out extensive simulations to evaluate the performance of the proposed heuristic and compare it with two other heuristics. One of those heuristics uses only dedicated resources while the other one uses both pools in a private-greedy manner.

The results indicate that the PCU concept of using public resources to augment private resources can lead to significant performance improvements both in terms of overall throughput obtainable from the computing utility and the service level compliance of the computing utility. Further, the results indicate that PCU performance depends on two major factors: characteristics of the public resources and characteristics of the workload. For instance, it can be noted that the performance gain from PCU increases if the job has fewer inter-job dependencies or relaxed deadlines. The performance of the PCU heuristic can be further improved by incorporating these parameters in the decision process.

One of the significant features of our PCU architecture is the minimal monitoring on the public resources. Because public resources are in plenty, this helps to keep the overhead low. It might be possible to selectively enable performance monitoring for high capacity public resources and increase the delivered performance levels even further.

This work is part of a bigger project Galaxy which is focusing on designing and implementing a complete PCU. We have introduced the Galaxy architecture and demonstrated the role of the proposed resource management heuristic in the context of the whole system.

References

[1] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, D. Werthimer, SETI@home: an experiment in public-resource computing, Comm. ACM 45 (11) (2002) 56–61.

[2] N. Andrande, F. Brasileiro, W. Cirne, M. Mowbray, Discouraging free riding in a peer-to-peer CPU-sharing grid, in: 13th IEEE International Symposium on High-Performance Distributed Computing, 2004.

[3] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. Pazel, J. Pershing, B. Rochwerger, Oceano—SLA based management of a computing utility, in: Proceedings of the Seventh IFIP/IEEE International Symposium on Integrated Network Management, 2001.

[4] M. Aron, P. Druschel, W. Zwaenepoel, Cluster reserves: a mechanism for resource management in cluster-based network servers, in: Proceedings of ACM SIGMETRICS, 2000.

[5] P. Bartley, M. Florian, P. Robillard, Scheduling with earliest start and due date constraints, Naval Res. Logist. Quart. 18 (1971) 511–519.

[6] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard, Web services architecture—W3C working draft, Technical Report, Web Services Working Group, World Wide Web Consortium, August 2003.

[7] R. Brayanrd, D. Kosic, A. Rodriguez, J. Chase, A. Vahdat, Opus: an overlay peer utility service, in: 15th International Conference on Open Architectures and Network Programming (OPENARCH), 2002.

[8] J.S. Chase, D.C. Anderson, P.N. Thakar, A.M. Vahdat, R.P. Doyle, Managing energy and server resources in hosting centers, in: 18th ACM Symposium on Operating Systems Principles, 2001.

[9] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, S. Tuecke, SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems, Lecture Notes in Computer Science, vol. 2537, Springer, Berlin, 2002, pp. 153–183.

[10] M.L. Dertouzos, A.K.-L. Mok, Multiprocessor on-line scheduling of hard-real-time tasks, IEEE Trans. Software Eng. 15 (12) (1989) 1497–1506.

[11] Ensim—web hosting software, ⟨http://www.ensim.com/⟩.

[12] M. Feldman, I.S.K. Lai, J. Chuang, Robust incentive techniques for peer-to-peer networks, in: ACM E-Commerce Conference (EC'04), 2004.

[13] I. Foster, C. Kesselman, The Grid: blueprint for a new computing infrastructure, Morgan Kaufmann, San Fransisco, CA, 1999.

[14] I. Foster, C. Kesselman, J. Nick, S. Tuecke, The physiology of the grid: an open grid services architecture for distributed systems integration, Technical Report, Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.

[15] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid: enabling scalable virtual organizations, Internat. J. Supercomput. Appl. 15 (3) (2001) 200–222.

[16] Y. Fu, A.M. Vahdat, Service level agreement based distributed resource allocation for streaming hosting systems, in: Seventh International Workshop on Web Content Caching and Distribution (WCW), 2002.

[17] M. Garey, D. Johnson, Two-processor scheduling with start-times and deadlines, SIAM J. Comput. 6 (1977) 416–426.

[18] M. Garey, D. Johnson, Computers and Intractability: A guide to the theory of NP-completeness, W H Freeman and Company, New York, 1979.

[19] C. Kenyon, G. Cheliotis, Creating services with hard guarantees from cycle harvesting resources, in: Proceedings of the Third IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03), 2003.

[20] E.L. Lawler, J.K. Lenstra, A.H.G.R. Kan, D.B. Shmoys, Handbooks in operations research and management science, vol. 4, Elsevier Science Publishers, 1993, pp. 445–522 (Chapter 9).

[21] A. Oram, Peer-to-Peer: harnessing the power of disruptive technologies, O'Reilly and Associates, Sebastopol, CA, 2001.

[22] Parsec: Parallel simulation environment for complex systems, ⟨http://pcl.cs.ucla.edu/projects/parsec/⟩.

[23] Planetlab: an open platform for developing, deploying and accessing planetary-scale services, ⟨http://www.planet-lab.org⟩.

[24] R. Raman, M. Livny, M. Solomon, Resource management through multilateral matchmaking, in: Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9), Pittsburgh, PA, 2000, pp. 290–291.

[25] R. Raman, M. Livny, M. Solomon, Policy driven heterogeneous resource co-allocation with gangmatching, in: Proceedings of the Twelfth IEEE International Symposium on High-Performance Distributed Computing, Seattle, WA, 2003.

[26] S. Ranjan, J. Rolia, E. Knightly, QoS driven server migraion for internet data centers, in: Proceedings of IWQoS 2002, 2002.

[27] A. Rowstron, P. Druschel, Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems, in: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, 2001, pp. 329–350.

[28] J. Sgall, On-line scheduling—a Survey, Springer, Berlin, 1997, pp. 196–231.

[29] K. Shen, H. Tang, T. Yang, L. Chu, Integrated resource management for cluster-based internet services, in: Proceedings of the Fifth USENIX Symposium on Operating Systems Design and Implementation (OSDI 02), Boston, MA, 2002, pp. 225–238.

[30] D. Thain, T. Tannenbaum, M. Livny, Distributed computing in practice: the condor experience, Concurrency and Computation: Practice and Experience 17 (2–4) (2005) 323–356.

Shah Asaduzzaman received his B.Sc. Engg. and M.Sc. Engg. degrees both in computer science and engineering from Bangladesh University of Engineering and Technology,s Dhaka, Bangladesh, in 1999 and 2002, respectively. Currently he is a Ph.D. student in the School of Computer Science in McGill University, Montreal, Canada. He is pursuing his research under supervision of Dr. Muthucumaru Maheswaran in the Advanced Networking Research Lab of this school. His current research interests include distributed systems architecture, scalable resource management, fault tolerance in public resource computing, distributed scheduling and peer-to-peer systems.

Muthucumaru Maheswaran is an Assistant Professor in the School of Computer Science at McGill University, Canada. From August 1998 to December 2002, he was an Assistant Professor in the Department of Computer Science at the University of Manitoba, Canada. In 1990, he received a B.Sc. degree in electrical and electronic engineering from the University of Peradeniya, Sri Lanka. He received an M.S. degree in electrical engineering in 1994 and a Ph.D. degree in electrical and computer engineering in 1998, both from the School of Electrical and Computer Engineering at Purdue University. His research interests include grid computing, peer-to-peer computing, trust modeling and management in large-scale networked systems, scalable resource management systems, custom computing for grid systems, and teaching and learning toolkits for computing networks. He has authored or coauthored more than 60 technical papers in these and related areas.