Heuristics for Scheduling Virtual Machines for Improving QoS in Public Computing Utilities

Shah Asaduzzaman and **Muthucumaru Maheswaran**
Advanced Networking Research Lab
School of Computer Science, McGill University
Montreal, QC H3A 2A7, Canada
{asad,maheswar}@cs.mcgill.ca

***Abstract*— *Computing utilities based on virtual machines (VMs) migrating among distributed computing resources are becoming increasingly practical and emerging as a viable infrastructure for outsourcing computer services. Here, we present a large scale distributed computing architecture named public public computing utility (PCU), where non-dedicated public resources are augmented with privately owned dedicated resources to obtain higher levels of quality of service (QoS) for compute intensive applications. Virtual machines are used to isolate foreign applications from the host systems and to facilitate seamless migrations. We have proposed several heuristics for scheduling VMs among the resources in PCU. Extensive simulations we performed indicate that the proposed heuristics are able to improve the QoS delivered by the PCU by utilizing the public and private resource combinations while reducing the VM migrations.*

Keywords: *Computing utility; Public resource computing; Resource management; Scheduling; Quality of service*

I. 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 the clients. In this paper, we are concerned about building CUs from *public* resources (i.e., resources that wish to contribute their computing, storage and network capacities without subjecting themselves to any contractual agreements) and we refer to them as *public computing utilities* (PCU).

Several large-scale network computing systems including *peer-to-peer* (P2P) file sharing systems such as Gnutella and volunteer computing systems such as SETI@home [1] have demonstrated the tremendous potential of using public resources. One of the desirable features of public resources is that they are virtually inexhaustible. For example, using the P2P network large numbers of resources can interconnect and form a public resource pool from which subsets can be discovered as required using DHT based discovery mechanisms [2]. Because the participation of resources is voluntary, high degrees of variability can be expected in the capacity. The mode of participation of the resources can vary widely from full dedication, partial dedication, to unknown dedication.

Unlike majority of the public resource based system, the PCUs are expected to host business critical applications with *quality of service* (QoS) requirements. Clients are provided with service quality based on the *service level agreements* (SLAs) signed with the PCU provider. The scheduling algorithms are designed such that highest level of conformance to the SLA is assured. The proposed architecture uses fully dedicated private resource pools to augment the capacities harnessed from public resources in order to deliver higher QoS guarantees to the applications.

We take a virtual machine based approach where application processes are encapsulated in virtual machines. The use of virtual machine is advantageous because they provide seamless mechanisms for job migration and monitoring with low residual dependencies.Virtual machines are becoming viable due to the advancements made in virtual machine monitors (VMMs) such as VMware [3] and Xen [4] for efficient management of VMs through the network. Checkpointing and migration are viable options for several useful applications. For stateless applications like web servers, the checkpointing cost is very low. But for stateful applications the cost of checkpointing depends on the volume of the state information the application needs to save for a restart. So our scheme will be suitable for the stateless or low-stateful applications.

We organized the rest of the paper as follows. Section II discusses some related literature on resource management of computing utilities and the use of virtual machines. Section III defines the system model we used and states the underlying assumptions. In Section IV, we present our proposed heuristics. In Section V, we demonstrate a thorough analysis of the heuristics using simulation results.

II. Related Work

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 [5] and Globus Grid [6]. 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 [7][8] with different optimization goals in different cases.

The Condor project [9] focuses on harvesting unused resources from heterogeneous public machines, but their resource management mainly emphasizes on discovery and co-allocation of resources through matchmaking and gangmatching. They do not support SLA driven QoS aware resource management on the public resource pool. *Cluster Computing on the Fly* (CCOF) [10] is a new P2P system that utilizes idle cycles from public domain resources to offer user QoS of different applications. But none of the above works have considered the conglomeration of public and private resources to enhance the QoS for applications.

Previous work that addresses the resource management problem on a similar settings can be found in [11], where the authors proposed scheduling heuristics for such settings and compared their performance with other classical scheduling algorithms. The major difference between [11] and this work is that the scheduler in [11] remains totally oblivious of the global system state, in order to keep the network overhead minimal. However, due to this oblivious nature of the system, there was a significant loss of the work done on public resources. It is important to investigate how the system state information can be gather with low overhead and how much benefit can be achived from better-informed decisions of the scheduler. The main goal of this paper is to trade-off the overheads and benefit and engineer the scheduling policies and system parameters for optimal performance.

Our proposed system uses virtual machines to encapsulate jobs to facilitate checkpointing and migration between different hosts. One big concern about using VMs in PCU is the cost of migrating VMs across different hosts. Nevertheless, there are several benefits of using this approach instead of migrating individual processes. Although process migration is much faster, several problems cripples its practicality. Residual dependency [12] (files and states that migrated processes leave behind on the previous hosts) and infrastructure compatibility [13] (e.g., missing libraries) are two notable problems. Several researches have focused on reducing the application down-time during a live migration [4] of virtual machines. Recently, heavily loaded web severs and game severs running on Xen has been live migrated with down time in the order of 100ms only.

III. Model and Assumptions

In our model, the PCU consists of a virtually infinite pool of non-dedicated public resources that are connected in a P2P overlay network and can be discovered based on resource attributes. At different locations within the larger overlay network, service providers maintain finite sized pools of dedicated resources that can be used by the PCU. Whena PCU client submits a job to the RMS, it defines resource requirement for the job per unit time. If the RMS agrees to provide the requirements encoded in an SLA, it launches the job encapsulated in a VM on an appropriate resource. Depending on the delivered capacity and SLA-compliance monitored by the virtual machine the RMS migrates the VM onto a new and more suitable resource, if necessary. To reduce the scheduling overhead, the RMS scheduler executes the scheduling rules at the end of each scheduling epoch of length δ. For the remainder of this paper, we consider the case where only one VM is allocated per job. However, for fault-tolerance purposes, a job can be duplicated and different instances of a job can execute on different VMs. In order to reduce the complexity of the RMS, we make several assumptions on the components of the PCU, which are explained in the rest of this section.

A. Resources

The private resources are assumed to be finite and homogeneous. That is the private resource pool has n resources with each having a steady processing capacity P_r in terms of MIPS (million instructions per second). The number of public resources is very large relative to n(infinite for this paper). However, each public resource is vulnerable and has a much higher probability of defaulting with regard to the task assigned to it. Their processing throughput is also variable between $P_{b_{min}}$ to $P_{b_{max}}$, where $P_{b_{min}} \leq P_{b_{max}} \leq P_r$

Because the public resources are inherently unreliable, the VMs that run on them checkpoint their execution state at the end of every epoch. In the simplest case, these checkpoints are stored in a remote location such that it could be retrieved if the VM fails at a future epoch and a new replica could be started. However, this approach can lead to performance bottlenecks and single points of failures. A distributed approach would be to store the checkpoints on a P2P storage system such as OceanStore [14].The detailed mechanisms for checkpointing, migrating, and restarting VMs is out of the scope of this paper and is a topic of future research.

B. Job and SLA

During submission, a job j requests through the SLA a resource share B_j per unit time, where we assume $B_j \leq P$. This assumption is reasonable, because if the client has a large application, it should be divided into several small independent subtasks. Then each subtask could be submitted separately and run on different resources in parallel. Other SLA parameters are the service ratio ρ and the monitoring time window T. The QoS that the SLA intends to guarantee is the minimum workload completed or the minimum resource capacity delivered over the whole monitoring window. The provider is compliant with the SLA if it can deliver total workload $V_j \geq \rho B_j T$, over the whole window, otherwise it is penalized. Also, the SLA defines the number of time windows t, for which it needs the resource share. The total workload of the job is therefore tB_jT.

C. Dissatisfaction metric

Because private resources are scarce, we need some priority scheme to rank the jobs competing for them. We define a metric called *unsatisfactory cost* (UC), which measures the degree of dissatisfaction of the job due to SLA incompliance. Higher value of UC prioritizes the jobs to get private resources.

Say, V_{jn} is the accumulated amount of resource-share the job j has received in the current SLA window till the end of nth epoch in the window. If v_{jn} is the amount of resource-time the job received in the nth epoch, V_{jn} is defined by –

$$\begin{aligned} V_{j0} &= 0 \\ V_{jn} &= V_{j(n-1)} + v_{jn} \end{aligned}$$

When K replicas of the job is running on K public resources asynchronously, one of them has the maximum progress and V_{jn} is computed as the resource-time given to this replica. If m of the K replicas fail during the nth epoch, m new replicas are started at the end of this epoch from the checkpoint with progress V_{jn}. If v_{jnk} is the amount of resource-time the kth replica of the job j received in the nth epoch, then V_{jn} is given by –

$$\begin{aligned} V_{j0} &= 0 \\ V_{jn} &= V_{j(n-1)} + \max_k \{v_{jnk}\} \end{aligned}$$

The *unsatisfactory factor* l_{jn} for job j at the end of nth epoch in current SLA window is defined as –

$$l_{jn} = \begin{cases} (1 - \frac{V_{jn}}{nB_j}) & \text{if } V_{jn} \le nB_j \\ 0 & \text{if } V_{jn} > nB_j \end{cases}$$

(CUC_j) of job j is then derived as –

$$CUC_j = (a_i^{l_{jn}} - 1)$$

Here $a_i \in (1, 2]$ is a real number assigned to the job according to the *static* priority that is determined by the subscription level of the client. Apparently for this range of a_j, always $CUC_j \in [0, 1]$. The higher the value of a_i, the more the CUC is sensitive to SLA deviation.

For each job, there is a parameter recording its previous service status, called *Historic Unsatisfactory Cost*(HUC_j), which is updated at the end of every SLA window according to the compliance during that window.

$$HUC_j = \begin{cases} \max(HUC_j - 1, 0) & \text{if CU compliant} \\ HUC_j + 1 & \text{if CU incompliant} \end{cases}$$

Next, we define a parameter *Total Unsatisfactory Cost*(TUC_j) which is used by job to compete resources: $TUC_j = CUC_j + HUC_j$.

IV. Resource Provisioning Heuristics

As mentioned earlier, resource provisioning refers to the task of assigning resources to jobs. In this section, we are proposing three heuristics for resource provisioning. These heuristics consider the private resources as the most important commodity and try to share them as much as possible among the jobs. The basis of all the three heuristic is to re-allocate the private resources to the jobs based on their progress at the end of each epoch. This overall scheme is illustrated in the Algorithm 1

Algorithm 1 Skeleton Scheduler

```
IDs of all the currently running jobs are stored in the Round Robin queue
for every epoch do
    insert the ID of all the newly arrived jobs into RR queue as long as length(RR queue) $\le$ Threshold; Discard rest of the new jobs
    ScheduleJobsInQueue(scheme)
    run scheduled jobs in appropriate resource for one epoch ($\delta$);
    for every job that finished execution do
        remove the job's ID from RR queue;
    end for
    update $CUC$, $HUC$ and $TUC$ for all jobs in the RR queue;
end for
```

In the simplest case of *Round Robin* scheduling, all the jobs retain equal priority and they time-share the private resources. That would be inefficient because jobs mapped on the public resources will have different progress levels. Therefore, an obvious improvement is to share the private resources among those jobs that are least satisfied. This approach is named *Ranked Round Robin*, because the jobs are ranked based on the dissatisfaction metric discussed in Section III-C. Because duplication or redundancy has proved to be a good way to combating the unreliability of the public resources, we can use this technique to improve the resource availability in the basic provisioning techniques. Hence, in the third approach, namely, *Ranked Round Robin with duplication*, each job is replicated by k different instances on different resources, when it is assigned a public resource. The algorithms for three different approaches are schematically shown in Algorithm 2.

The number of replica's is, maintained to be k in every epoch by re-launching a new replica for every failed replica. An estimate of k can be found from reliability theory [15]. If the failure process is Poisson, mean contiguous available time of a resource is $\frac{1}{\lambda}$ and mean recovery time is $\frac{1}{\mu}$, then the minimum number of replicas (k) required to keep at least 1 replica alive all the time can be derived as,

$$k = \frac{log(1-\alpha)}{log(\lambda) - log(\lambda + \mu)}$$

where, α is the desired level of confidence. For example, if we want, with 95% probability, that 1 resource be available all the time, and we have MTTF $\frac{1}{\lambda} = 5$ minutes and recovery time $\frac{1}{\mu} = 2$ minutes, then k turns out to be 2.39.

Algorithm 2 ScheduleJobsInQueue(*scheme*)

```
if scheme = Round Robin then
    move the IDs for $p$ jobs running on the private resource to end of RR queue;
    assign private resource to the jobs of first $p$ IDs in RR queue;
    assign public resource to rest of the jobs in RR queue;
else if scheme = Ranked Round Robin then
    sort all the job IDs by the job's TUC in RR queue;
    assign private resource to the jobs for first $p$ IDs in RR queue;
    assign public resource to the jobs for rest of the IDs in RR queue;
else if scheme = Ranked Round Robin + k duplication then
    sort all the job IDs by the job's TUC in RR queue;
    assign private resource to the jobs for first $p$ IDs in RR queue;
    for all the jobs for rest of the IDs in RR queue launch $k$ replicas of each of them on $k$ public resources;
end if
```

V. Performance Analysis

In this section, we evaluate the performance of our proposed heuristics through simulation studies. We developed a simulation model using C++ based on the PCU architecture described earlier in Section III. We studied the system for a wide range of parameter values and then chose the values for realistic performance gain. The basic model parameters are described below in Section V-A. We tried to measure the performance according to several performance criteria, mainly on the two different axes - the QoS achieved from using the heuristics and the overheads while delivering the QoS. The results are analyzed in Section V-B.

A. Simulation Model

The PCU consists of a homogeneous pool of dedicated or private resources that do not have any performance deviation or failure. We used a pool of 100 dedicated resources all having the same capacity. We compared our heuristics with the benchmark system having 100 private resources only and running a M/M/m/m queuing discipline. The public resources in PCU are gathered on ad-hoc basis from a wide area network like Internet. Although the number of public resource is virtually unlimited, to put a practical limit on this availability, we set a threshold in the RMS for maximum number of concurrent jobs, which is typically 10 times the number of private resources. The public resources show wide variation in their performance and also they may fail any time. In our experiments the failure hazard rate (instantaneous probability of failure) was 0.3 times in each 10 second epochs.

Jobs arrive in the system in a Poisson process. Each job has a random workload that takes 100 epochs on average if run on a dedicated machine. Each job specify their resource bandwidth requirement (in terms of resource-seconds) in the SLA. We assumed that this bandwidth never exceeds the capacity of the private resources. When a job needs a public resource, we assume that it is always possible find a public resource that matches a job's bandwidth requirement, because public resources are in plenty. However the performance actually delivered by public resources always vary from their nominal bandwidth capacity (with a Normal distribution around the nominal value in our experiments).

The epoch δ and the SLA monitoring window T were chosen to be 10 and 100 seconds, respectively. The service ratio ρ was assumed to be 0.8. According to the algorithms, the jobs (wrapped in VMs) need to migrate from one machine to the other and the cost of migration was chosen to be 1 second of downtime compared to 60-300 ms for Xen [4]. For stateful applications, it is not possible to overlap the migration with the job execution, so, we assume that for the whole migration time the job execution was down.

Although we did not explicitly model the network topology of the resources for simulation, the number of migrations, the down time of job execution due them and their effect on total job execution time is measured. These statistics well reflect the communication overhead on the PCU due to checkpointing and migration. As for the public rsources, we assume that gathering them from Internet does not incur any financial cost for the PCU.

B. Results

Hypothetically, if we have unlimited supply of public resources, the throughput of the system keeps increasing almost linearly with the given workload without any limit. But in practice the supply will be limited, therefore we choose some threshold on the number of concurrent jobs running in the system. The result is depicted in Figure 1. For the same threshold, the ranked versions of the algorithm shows much higher maximum throughput than the pure round robin. We also observe that the throughput of PCU is much higher than a baseline system of 100 private mahcines only. This justifies the use of public resources in addition to the expensive dedicated private machines. However, PCU throughput is is much less than a system having all 500 dedicated resources. This lack in performance is due to the variability and failure of the public resources. Because private resources are much more expensive than public resources, our proposed system with a combination of both will be preferable.

With the proposed ranked round robin algorithms the system tries to improve QoS of resource allocation by minimizing the violation of SLA specified bandwidth needs for the jobs. Still due to the variability of public resources, some violation of SLA is unavoidable, especially at high loads. If the system earns revenue in proportion with the amount of work it delivers, it should be penalized in proportion with the deviation

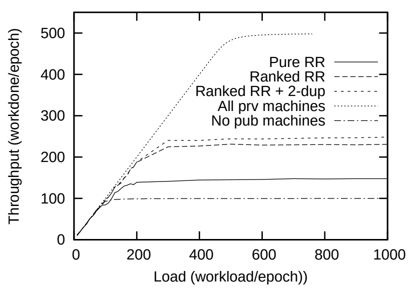

Fig. 1. Throughput variation with load for different algorithms (Threshold=500)

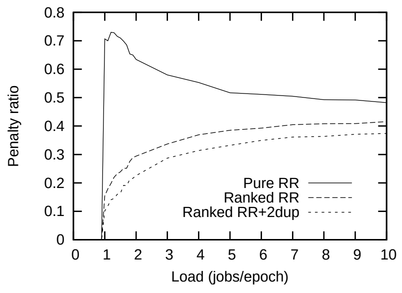

Fig. 2. Penalty (deviation) per unit work done

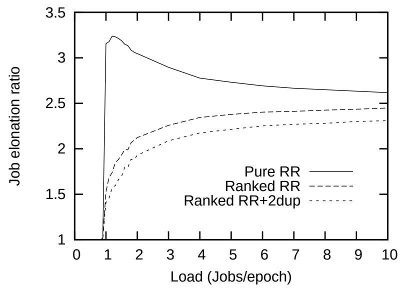

Fig. 3. Job elongation ratio for different algorithms

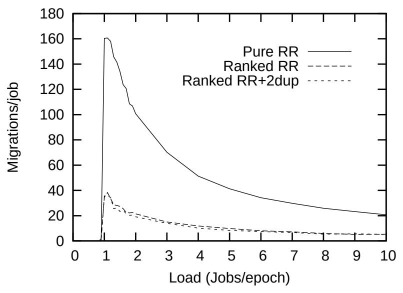

Fig. 4. Number of preemptive migrations per job (between private and public pool)

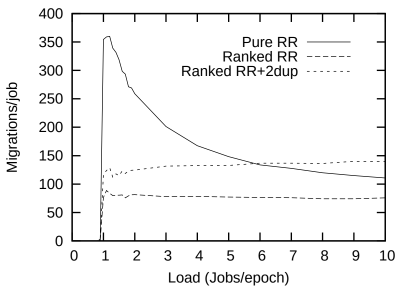

Fig. 5. Number of migrations per job due to failure

from SLA. Figure 2 shows the penalty incurred due to SLA violation for each unit of computational work done by PCU. Here also, the penalties are much less in the ranked version of the algorithm. This is because the ranked algorithm always prioritize the starving jobs for running on the private machines. Replication also serves for reducing the penalty because of two reasons - on one hand replication increases reliability and reduces the amount of failure; on the other hand we always take the maximally progressed replica for computing the SLA deviation.

In all the three algorithms, the VMs holding the jobs need to migrate between different machines. In the pure round robin, migration rate is high because it re-allocates the private resources to a different subset of jobs every epoch, in purely a round robin manner. In the ranked version, a job may be evicted from private resource only if another job with higher dissatisfaction is running on a public resource. Figure 4 shows the average number of preemptive migrations between private and public resource pools. In case of pure round robin, there is a sharp rise in the number of migration, when the number of jobs in the system is exactly twice the number of dedicated resources. In this case each job needs to migrate every epoch. The VMs also need to migrate if a public resource fails. The rate of migration of this type is obviously higher when replication is used (Figure 5). One interesting side effect of migration is that at each migration a new resource is contracted. Normally public resources may show some aging effect in their behavior from the time they are allocated. Frequent migration would refresh this aging process and hence increase the performance. Detail study of this refreshing effect is, however, out of the scope of this paper.

Due to migrations and also due to capacity-variation and failure of public resources, the jobs total run-time gets elongated with respect to the ideal case where each job alone is allocated a dedicated resource to execute until completino. Figure 3 shows this elongation effect for the 3 algorithms. Here also, the elongation is minimum for the duplicated ranked version of the algorithm. The elongation can be a factor of 2 in the worst case of very high loads.

From the above comparisons, it is apparent that the ranked round robin algorithm with duplication is the best choice among the 3 versions of the algorithms in all respects. Also, the gain in throughput justifies the use of additional public resources as long as they are inexpensive.

The performance of the algorithms also depends on different other environment parameters like number of dedicated machines, the failure characteristics of the public machines, the cost of migration etc. The following part of the section discusses the effect of these parameters on the duplicated-ranked round robin algorithm.

Figure 6 shows the effect of the number of private resources on elongation factor. Obviously for the same load, increasing the number of private resources reduces both SLA violation and elongation. In the best case when we have enough private resource to hold all the jobs, no penalty and elongation is incurred.

The behavior of the public resources, especially their failure rate affects the QoS factors (Figure 7). It shows that elongation increases super-linearly with the error

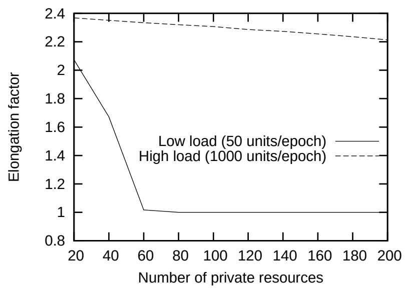

Fig. 6. Elongation factor for different number of private resources

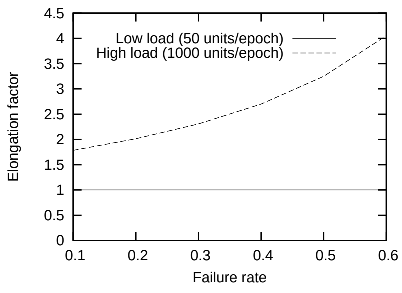

Fig. 7. Elongation factor for different failure rates of public machines

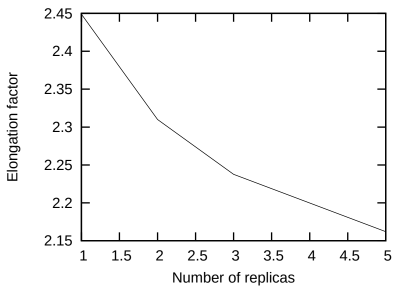

Fig. 8. Elongation factor for increasing number of replicas

rate of the public resources.

In our third version of the algorithm, we used replication for improving reliability of public resources. The replication of course multiplies the number of public resource requirements linearly. In Figure 8 we tried to compare the QoS achievements for increasing number of replications. The results show that the improvement in QoS is sub-linear with increasing number of replicas. So, for an appropriate cost of replication there will be an optimal number of replicas. Also, we have stated from reliability analysis in Section IV that a replication factor of 2 is adequate to have desired reliability from public resources.

VI. Conclusion

In this paper, we proposed an architecture for PCUs that augments finite number of private resources to public resources to improve the SLA compliance levels. Our architecture uses VMs to implement "time-sharing" over the resources. The objective of the time-sharing is to mask the unreliability of public resources by using them for fixed durations and saving the job progresses periodically to facilitate restarts. The simulation results presented here show that time multiplexing the resources can result in significant improvements for system throughputs and job QoS such as elongation and drop rates. Further, the number of job migrations can be reduced by incorporating "conditional" migrations that only move jobs that are highly likely to cause SLA violations.

Although the proposed architecture is practical given the recent advancements in VMs, the issues related to checkpointing and migration overhead should be investigated further to improve deployment. Devising an efficient scheme for checkpointing and migration, whether distributed, peer-based or centralized, is crucial to reduce this overhead. An overall cost-benefit analysis of the messaging and financial overhead of using public resources will shed some more lights on establishing a complete PCU system.

References

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

[2] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," in *IFIP/ACM Intl. Conf. on Distributed Systems Platforms (Middleware)*, Nov. 2001, pp. 329–350.

[3] J. Sugerman, G. Venkitachalam, and B.-H. Lim, "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor," in *USENIX'04*, Jun. 2004, pp. 1–14.

[4] 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 Symp. on Networked Systems Design and Implementation (NSDI'05)*, May 2005.

[5] 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 Intl. Symp. on Integrated Network Management*, May. 2001, pp. 855–868.

[6] 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," *Springer LNCS*, vol. 2537, pp. 153–183, Jul. 2002.

[7] 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 SOSP*, Oct. 2001, pp. 103–116.

[8] S. Sranjan, J. Rolia, H. Fu, and E. W. Knightly, "QoS Driven Server Migraion for Internet Data Centers," in *10th IWQoS*, May. 2002, pp. 3–12.

[9] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience," *Concurrency and Computation: Practice and Experience*, vol. 17, pp. 323–356, 2005.

[10] V. Lo, D. Zappala, D. Zhou, Y. Liu, and S. Zhao, "Cluster Computing on the Fly: P2P Scheduling of Idle Cycles in the Internet," in *3rd Intl. Workshop on Peer-to-Peer Systems*, Feb. 2004, pp. 66–73.

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

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

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

[14] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An Architecture for Global-Scale Persistent Storage," in *9th intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, Nov. 2000, pp. 190–201.

[15] S. M. Ross, *Introduction to Probability Models*, 5th ed. Academic Press Inc., 1993, pp. 410–546.