Enabling Interactive Jobs in Virtualized Data Centers (Extended Abstract)

John Paul Walters, Bhagyashree Bantwal, and Vipin Chaudhary
Department of Computer Science and Engineering
University at Buffalo, The State University of New York
{waltersj, bbantwal, vipin}@buffalo.edu

***Abstract*— Existing batch schedulers are incapable of adequately addressing the need for immediate access to resources for interactive jobs. In this paper we describe our virtual machine-centric scheduler, UBIS, that facilitates both high priority interactive jobs and traditional batch jobs. UBIS is based on preemptable job scheduling which enables rapid resource provisioning for interactive jobs, and easy job restarts for preempted jobs. Using simulation we characterize the impact of interactivity on the data center and develop a novel set of scheduling criteria that emphasizes near-immediate resource allocation for interactive jobs. We show that our new scheduling criteria not only improves resource allocation for interactive jobs, but also improves resource utilization and response times for batch jobs up to 500%. Our system will serve as a valuable tool to enable interactive jobs to execute efficiently within a traditional batch scheduling environment.**

I. Introduction

With the current generation of supercomputers now achieving petaflop-level performance, applications are placing even greater demands on the underlying hardware. Interactivity will play a key role in the next generation of supercomputers [1]. Where traditional supercomputing has focused mainly on batch processing with offline visualization/interactivity, interactive computing seeks to provide users with the resources to perform both the computation and interaction simultaneously. We argue that this will require a new strategy towards dealing with supercomputing resources.

Interactivity requires advances in scheduling, process management, and data management. New scheduling policies that are capable of allocating and reallocating interactive resources on-demand will be especially critical. This will necessitate a change in data center and cluster process management as batch jobs must be preemptable in order to ensure adequate resources for the interactive jobs. Data management will also be critical, as both preempted (checkpointed) batch jobs and the data generated by the interactive jobs will have to be efficiently managed.

Thus far interactive supercomputing has received only modest attention in the academic community, while existing batch schedulers seemingly treat interactive jobs as mere afterthoughts. A key to this work is the use of low overhead virtualization strategies in order to provide the building blocks of computation migration and checkpointing. These features will be especially critical to enabling interactive computing while also allowing preempted jobs to be resumed with a minimal loss of computation.

This research was supported in part by NSF IGERT grant 9987598, MEDC/Michigan Life Science Corridor, and NYSTAR.

However the use of virtualization also necessitates an effective provisioning system. The provisioning framework should cooperate with the scheduler in order to anticipate job requirements without dedicating unneeded resources. Many of today's VM provisioning systems naively swap operating systems without considering the resource trends. This leads to wasteful VM swapping and lower cluster utilization. While the focus of this paper is on the scheduling, we note that provisioning will prove a crucial component of any VM-based preemptive scheduler.

We focus this paper on the particular scheduling implications of supporting interactive supercomputing. In doing so, we introduce UBIS, the University at Buffalo Interactive Scheduler. We describe our interactivity framework, and through a preliminary performance evaluation of the scheduler, demonstrate that interactivity can be achieved with minimal cost to traditional batch jobs.

The remainder of this abstract is organized as follows: in Section II we describe the existing work in interactive scheduling, in Section III we present the framework for combining our interactive scheduler with VM swapping and preemption. In Section IV we provide preliminary results of the effectiveness of our interactive scheduler through simulation, and in Section V we describe the future directions and preliminary conclusions that can be drawn from our work.

II. Related Work

The scheduling of tasks on multiprocessor systems has been well-studied. Approximate algorithms and heuristics have been proposed that attempt to minimize the makespan of the cluster, or completion of the last job. Most scheduling does not consider preemption, or when it does, may neglect the potential for job starvation in on-line systems. In this section we briefly discuss the major approximation algorithms that have been proposed,.

To date, most scheduling has focused on the non-preemptive variety where jobs are assigned to nodes (or parallel machines) and allowed to run to completion. This has resulted in numerous approximation algorithms. Paletta and Pietramala present a technique that combines partial scheduling solutions into a single solution [12]. Genetic algorithms have been particularly well studied in recent scheduling work [2], [3], [6], [11], [17]–[19]. Other strategies including linear programming, insertion-based heuristics, simulated annealing, and tabu search have also been studied [4], [10]. Jin et al. presents a comparison of many different scheduling approximation techniques and shows that most existing techniques are better suited to offline scheduling [9].

Researchers have also considered preemtable scheduling [7]. A common strategy is to convert the problem into a non-preemptive scheduling problem [13]. This typically leads to an improved approximation factor, but is not directly applicable to our work.

Sotomayor et al. study the issue of virtualization overhead [14]. In particular, they argue that when integrating virtualization into a grid scheduler, the overheads of instantiation, and deployment must be accounted for correctly. Our preemptive scheduler accounts for such overhead, and includes the overhead of checkpointing and migrating jobs.

Further work by Sotomayor et al. has examined the use of a lease-based architecture to negotiate advance reservation requests [15]. Rather than basing scheduling decisions on specific jobs, decisions are based on leases that describe both the hardware and software requirements of an applications. The use of advance reservations has previously been studied by Foster et al. for grid scheduling [5]. Like the UBIS, Sotomayor et al. use virtualization in order to provide the needed checkpoints/restore functionality. Our preemption strategy, however, is not based on the idea of advance reservations to support job interactivity. We make no assumptions as to when an interactive job will enter the queue and require immediate resources.

III. System Development

We have incorporated the scheduling algorithms into a prototype system capable of evaluating our models and assumptions in a production-like environment. To do so we use a virtualized cluster which is interconnected via a network with a server that handles the node provisioning. The implementation of this framework consists of a resource manager, a scheduler, scalable OS distribution and a VM invocation framework, and is based on the Torque batch scheduler.

A. Resource Manager

The resource manager is responsible for providing control over batch jobs and distributed compute nodes. It also maintains a registry of all cluster nodes and any attributes that impact the node provisioning decisions of the scheduler. This is done by the MOM (machine oriented miniserver) daemon, which periodically probes the cluster nodes and obtains a node's status from the daemons on the cluster nodes.

The resource manager typically represents a node by a set of attributes consisting of the IP, status (free, down), base operating system, etc. The base operating system attribute indicates the operating system used by the physical server. In our case, this is the operating system used by the hypervisor. We therefore introduced a new attribute, VM_{os} which indicates the guest virtual machine's operating system. This node attribute is now a dynamic one and needs to be changed as the VMs are invoked.

To allow the virtualized guest to act as a traditional Torque client, the MOM daemon is added to the guest. This effectively creates an independent client from the virtual machine, and allows the scheduler to treat an individual VM as a full client. It also has the added impact of forcing the Torque node registry to become dynamic due to changing virtual machines.

By incorporating these attributes the cluster can be represented as a pool of virtual resources. Thus, any job that requires a unique set of resources (e.g. an alternative OS) can be serviced from this pool. In this way, the cluster is limited only by its available hardware, not software. Any user may request unique software which can be rapidly provisioned and made available for the duration of the job.

B. The Scheduler

The scheduler is responsible for the scheduling of jobs that are placed into Torque's job queue and allows for the use of a cluster's idle resources in order to satisfy a job's requirements. When a job is placed into the job queue an initial priority is calculated from the job attributes and the priority is analyzed by the resource manager before being placed into the scheduler queue. Our approach is shown in Equation 1. In this strategy it is expected that the lowest priority jobs will be preempted first.

$$\begin{aligned} P_j = {} & w_1 \cdot (Target - Usage) + w_2 \cdot Q_{time} + \\ & w_3 \cdot (N_{availableOS} - N_{reqOS}) + w_4 \cdot P_n + w_5 \cdot P_t + \\ & w_6 \cdot (N_{free} - (N_{reqOS} - N_{availableOS})) + w_7 \cdot I \end{aligned} \tag{1}$$

Where $Target$ is a user's expected usage (e.g. monthly, weekly); $Usage$ is a user's actual usage; Q_{time} is the time a job has spent in the queue; $N_{availableOS}$ is the number of nodes currently available with a given operating system; N_{reqOS} is the number of nodes of a certain operating system needed by a job; P_n is the number of times this job has been preempted; P_t is the total time the job has been preempted; N_{free} is the total number of free nodes of any operating system; I is 1 if the job is interactive, else 0; $w_1, w_2, \ldots, w_6$ are weighting factors that are empirically determined; and w_7 is the weighting factor to elevate interactive jobs.

By accounting for the operating systems in use as well as the free resources, we are able to improve on the scheduling by reducing the number of operating system swaps. The term $(Target - Usage)$ simply describes a user's current usage. If the user has not been particularly active, then the quantity will be higher than a user who has exceeded his target usage. The second quantity Q_{time} simply accounts for the time a job has spent in the queue prior to being scheduled. As a job waits in the queue, its Q_{time} will increase thereby increasing the priority. The quantity $(N_{availableOS} - N_{reqOS})$ allows us to account for the current distribution of operating systems within the cluster. A negative value indicates that the user is requesting nodes that will require provisioning. This decreases the job's priority. In order to prevent jobs from starving due to preemption, we increase the value of P_n as the number of preemptions increase, thereby raising the priority of the more frequently preempted jobs. Similarly, the value of P_t increases with the total time that a job spends in the preempted state. We track both the number of preemptions and the time spent preempted in order to account for jobs that may be preempted multiple times, but for short durations. Intuitively, we expect that greater P_t with lower P_n is preferrable. However, this assumption must be validated under varying workloads and job characteristics. The terms $w_1, \ldots, w_6$ are weights that are empirically determined with simulation. Their values will depend on the various scheduling policies that we are evaluating, and will differ between policies. Some values of w_i lead to greedy or best fit algorithms. The term I is either 0 or 1 to indicate whether or not the job is interactive. Its weight, w_7 which will also be experimentally determined, could be much larger than the others to immediately elevate an interactive job's priority.

Backfill has often been used to allow for out-of-order job execution, and has been shown to improve utilization by as much as 20% [8]. The idea is to schedule jobs that can make use of the currently available resources rather than allowing those resources go unused. The scheduler protects the highest priority job's start time by creating a job reservation to reserve the needed resources at the appropriate time. It can then start any job that will not interfere with this reservation. Our approach uses the Torque backfill approach to improve batch job scheduling.

C. VM Provisioning Framework

Once the scheduler identifies the idle nodes that can be provisioned for new jobs, the required OS images are provided to the VM invocation framework. A balanced binary tree is used to query, forward, and retrieve OS images. All OS images are initially stored at the head node. In the event of a request for a particular OS image, the image is sent via the node's children to the node requesting the resource. All nodes along the path including the final node maintain information of the node which has been given that OS image, in a routing table. The information in the routing table is then reused when another request for the same OS image occurs.

Once the idle node receives the OS images, configurations, and swap files, the VM is instantiated and the MOM daemon is started on this VM. Once the VM is instantiated the control is passed to the scheduler enabling the node to service jobs allocated by the scheduler.

D. Job Preemption

In the event that resources are unavailable for a high priority job, one or more jobs must be preempted in order to enable interactivity. There are three major facets to job preemption that must be addressed: scheduling, checkpointing, and restarting jobs. We address the challenges of each in the paragraphs to follow.

The first and the most challenging aspect that must be considered is the scheduling and selection of preemptable jobs. In a data center with thousands of nodes and jobs, making good preemption choices will be critical to improving the response time of the cluster. Our current strategy is to preempt the lowest priority jobs as determined by Equation 1.

Once a preemptable job has been identified, a checkpoint is taken in order to save the state of the preempted jobs such that they can be restarted at a later time without a substantial loss of computation. Hypervisors and other virtualization solutions typically provide a mechanism to save the memory footprint of an individual VM. In addition to the memory footprint, however, the file system must also be checkpointed. Finally, if a job is executing in a distributed manner, such as via MPI, the distributed state must also be synchronized before checkpointing. In our previous work we have demonstrated a novel VM-aware MPI implementation [16]. A similar approach is used in this case.

Finally, once the interactive job has completed, the preempted jobs should be restarted. In the simplest case, the original jobs are simply restarted on the machines from which they were preempted. This can be done with little overhead, as all of the data is already present on the necessary nodes. However, if additional resources become available before the interactive job has completed, it may be advantageous to restart the preempted jobs on alternate nodes. This will depend on the overhead of transferring the needed data to new nodes (as checkpoints, particularly file systems can be quite large). Because the focus of this abstract is primarily on our scheduler, we do not include this work within the scope of the abstract.

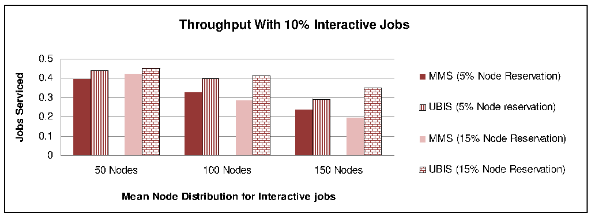
E. The System Architecture

From the user's perspective, the system consists of three major components: (i) a scheduler/resource manager (ii) a VM repository/provisioning framework, and (iii) a checkpoint/migration infrastructure.

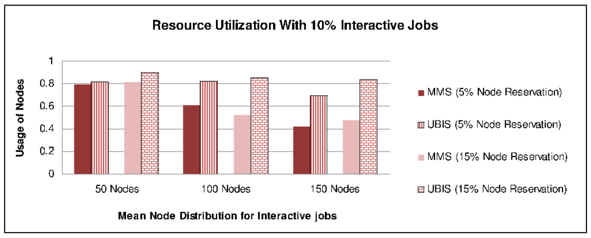
Jobs are submitted to the resource manager where the scheduler determines their priorities and inserts the jobs into the job queue. Depending on the type of job it may be handled in one of two ways. If the job is a standard batch job, its job requirements are collected and the job is treated as a typical batch job. A batch job will not ordinarily preempt another batch job. Once the job is scheduled for execution, it will either run directly within one or more existing VMs (preferred to reduce provisioning time), or the resource manager will provision a set of VMs for its execution. The job will either run to completion or will be preempted by an interactive job during computation (described below).

In the event that an interactive job is submitted, its priority is elevated. If the needed resources are already idle, the interactive job will immediately be scheduled onto the free nodes. As in the case of the batch job, the nodes may need provisioning if they are not already running the required VM. If the resources are not idle, one or more jobs may be preempted in order to make room for the interactive job.

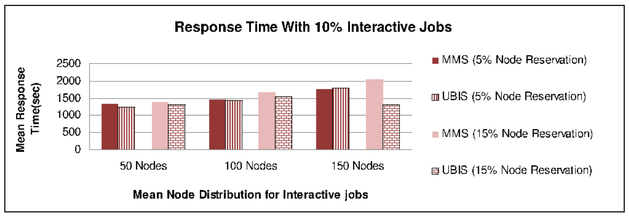
Preempting a job is a multi-step process. First, the running job (to be preempted) must be checkpointed. If the job is a single node (non-MPI) job, the checkpointing process is



(a) Throughput, 10% interactivity

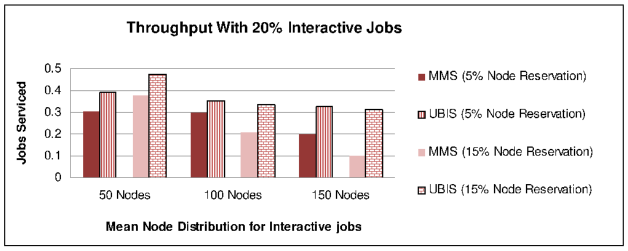


(b) Resource utilization, 10% interactivity

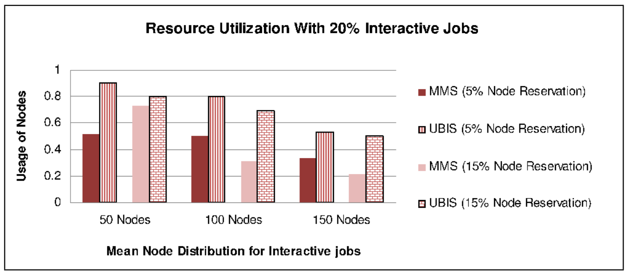


(c) Response time, 10% interactive jobs

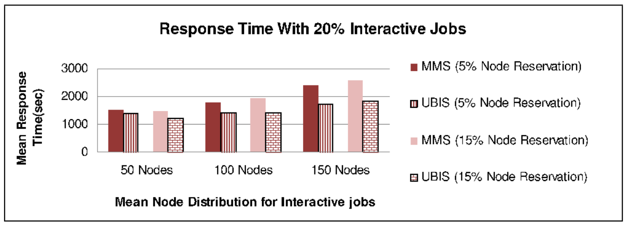
Fig. 1. 10% job interactivity.



(a) Throughput, 20% interactivity



(b) Resource utilization, 20% interactivity



(c) Response time, 20% interactive jobs

Fig. 2. 20% job interactivity.

simple and can be handled within the virtual machine itself. If the job is composed of multiple nodes communicating via MPI, the nodes must first quiesce all messaging channels using a custom virtual machine-aware MPI library [16] before checkpointing the VMs. After checkpointing the VMs, the provisioning framework will provision the VMs as-needed and the interactive job will run. After the interactive job completes the preempted job will continue execution.

IV. Experimental Results

Here we present the preliminary results of our scheduling policy, obtained through simulation. In the Figures 1-3 we demonstrate the impact of our interactive scheduling policy compared against a modified Maui FCFS scheduling policy (MMS in figures). Our modified Maui scheduler includes only those changes necessary to support operating system swapping, but does not change in any way its scheduling decisions.

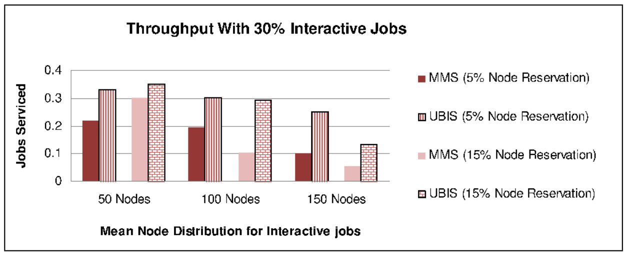
We use trace data gathered from the University at Buffalo's Center for Computational Research (CCR) and simulate for 100,000 jobs. Operating systems are assigned based on a random function. Our trace data is assumed to run on a cluster of 1056 nodes (the size of the CCR cluster). Interactive jobs are randomly injected into the trace data with the size of the interactive jobs governed by a Poisson distribution with means of 50, 100, and 150 nodes. The execution time of the injected interactive jobs is between 10 and 120 minutes, according to a uniform distribution.

The figures show the results of our scheduling policy compared against the Maui scheduler with 10, 20, and 30% of the cluster devoted to interactive jobs, with the trace data from CCR. The weights chosen from Equation 1 are $w_1 = 0.07, w_2 = 0.13, w_3 = 0.1, w_4 = 0.1, w_5 = 0.16, w_6 = 0.2, w_7 = 0.24$. Again, these weights are preliminary. For Maui scheduling, a reservation system is used to gather nodes for an interactive job as-needed. This is the standard behavior for the Maui scheduler.

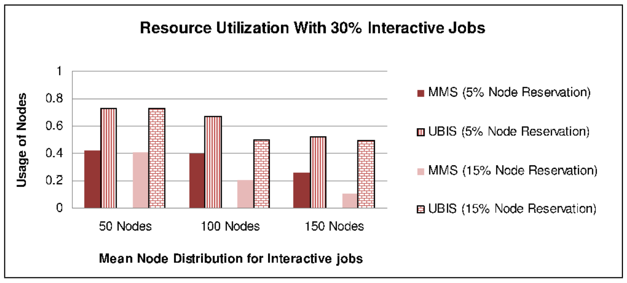
For all results, the operating system distribution is divided between two OSs with a $50/50$ distribution. We simulate our scheduling policy to allow for 5% and 15% of the cluster nodes devoted to interactive jobs. Thus, when we indicate 10, 20, or 30% of the jobs are interactive with 5 or 15% of nodes devoted to interactive jobs, we mean that of all 100,000 simulated jobs, 10, 20, or 30% will be interactive. Further, at any given time, only 5 or 15% of the cluster nodes may be devoted to interactive jobs. Any remaining interactive jobs will block waiting for free resources.

From Figures 1, 2, and 3 we can see that the introduction of interactivity generally results in an improvement on resource utilization, response time, and throughput. We demonstrate consistent improvement between the UBIS scheduler and the Maui scheduler. This is to be expected as the Maui reservation system blocks for available nodes, preventing any new jobs from executing until the high priority job receives its resource requirements.

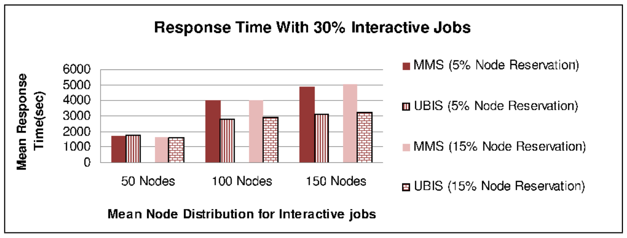
The only exception to this is Figure 3(c) with interactive job sizes of mean 50 nodes and 5% node reservation. Even in



(a) Throughput, 30% interactivity



(b) Resource utilization, 30% interactivity



(c) Response time, 30% interactive jobs

Fig. 3. 30% job interactivity.

this case, however, the overhead is only 2.8%. As one would expect, the response time increases with increasing interactive job sizes as well as with larger percentages of the cluster allowed for interactivity. This is typical behavior of any scheduler. Nevertheless, we consistently outperform the Maui scheduler in all metrics, even when 30% of the simulated jobs are interactive. Further, the performance increasingly improves relative to the modified Maui scheduler as the number of interactive jobs increases from 10% to 30%, resulting in as much as **500%** improvement for resource utilization, as we show in Figure 3(b).

V. Conclusions and Future Work

This project presents the first step in enabling a new paradigm in traditional supercomputers from being purely batch job based to allowing interactive jobs. We have shown that interactivity can be introduced into existing schedulers with minimal detrimental impact. Further, we have shown that a performance of up to **500%** improvement has been shown in real trace data. This opens up doors to various non-traditional applications that are limited by the time when the results are needed, e.g., most analytics. The outcomes of this project will also enable data centers or providers to efficiently utilize their infrastructure, thereby reducing cost and increasing profit.

We will continue to work towards improving the weights described in our priority equation (Equation 1). We believe that through extensive modeling more effective coefficients can be found that will further reduce the impact of interactivity on overall cluster resources. We will then continue to integrate our results into a working system for deployment.

References

[1] R.E. Bryant. Data-Intensive Supercomputing: The Case for DISC. Technical Report CMU-CS-07-128, 2007.

[2] S.-C. Cheng and Y.-M. Huang. Scheduling multi-processor tasks with resource and timing constraints using genetic algorithm. *Computational Intelligence in Robotics and Automation, 2003. Proceedings. 2003 IEEE International Symposium on*, 2:624–629 vol.2, 16-20 July 2003.

[3] R.C. Corrêa, A. Ferreira, and P. Rebreyend. Scheduling Multiprocessor Tasks with Genetic Algorithms. *IEEE Trans. Parallel Distrib. Syst.*, 10(8):825–837, 1999.

[4] E. Onbaşçioglu and L. Özdamar. Optimization of Data Distribution and Processor Allocation Problem Using Simulated Annealing. *J. Supercomput.*, 25(3):237–253, 2003.

[5] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and Alain Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *Proceedings of the* 7^{th} *International Workshop on Quality of Service*. IEEE Computer Society, 1999.

[6] E. S. H. Hou, N. Ansari, and H. Ren. A Genetic Algorithm for Multiprocessor Scheduling. *IEEE Trans. Parallel Distrib. Syst.*, 5(2):113–120, 1994.

[7] J. Błażewicz and M. Drozdowski and P. Formanowicz and W. Kubiak and G. Schmidt. Scheduling preemptable tasks on parallel processors with limited availability. *Parallel Comput.*, 26(9):1195–1211, 2000.

[8] D. B. Jackson, Q. Snell, and M. J. Clement. Core Algorithms of the Maui Scheduler. In *JSSPP '01: Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 87–102, London, UK, 2001. Springer-Verlag.

[9] S. Jin, G. Schiavone, and D. Turgut. A Performance Study of Multiprocessor Task Scheduling Algorithms. *J. Supercomput.*, 43(1):77–97, 2008.

[10] J. Jungwattanakit, M. Reodecha, P. Chaovalitwongse, and F. Werner. A Comparison of Scheduling Algorithms for Flexible Flow Shop Problems with Unrelated Parallel Machines, Setup Times, and Dual Criteria. *Comput. Oper. Res.*, In press 2007.

[11] C. Oguz and M.F. Ercan. A genetic algorithm for multilayer multiprocessor task scheduling. *TENCON 2004. 2004 IEEE Region 10 Conference*, B:168–170 Vol. 2, 2004.

[12] G. Paletta and P. Pietramala. A New Approximation Algorithm for the Nonpreemptive Scheduling of Independent Jobs on Identical Parallel Processors. *SIAM Journal on Discrete Mathematics*, 21(2):313–328, 2007.

[13] U. Schwiegelshohn. Preemptive Weighted Completion Time Scheduling of Parallel Jobs. *SIAM Journal on Computing*, 33(6):1280–1308, 2004.

[14] B. Sotomayor, K. Keahey, and I. Foster. Overhead matters: A model for virtual resource management. In *VTDC '06: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*. IEEE Computer Society, 2006.

[15] B. Sotomayor, K. Keahey, and I. Foster. Combining batch execution and leasing using virtual machines. In *Proceedings of HPDC '08: the International Symposium on High Performance Distributed Computing*. ACM Press, 2008.

[16] J. P. Walters and V. Chaudhary. FT-OpenVZ: A Virtualized Approach to Fault-Tolerance in Distributed Systems. In *In Proceedings of ISCA PDCS*, pages 85–90, 2007.

[17] A.S. Wu, H. Yu, S. Jin, and K.-C. Lin. An Incremental Genetic Algorithm Approach to Multiprocessor Scheduling. *IEEE Trans. Parallel Distrib. Syst.*, 15(9):824–834, 2004. Member-Guy Schiavone.

[18] C. Zhang, P. Li, Z. Guan, and Y. Rao. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research*, 34(11):3229–3242, November 2007.

[19] A.Y. Zomaya, C. Ward, and B. Macey. Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues. *IEEE Trans. Parallel Distrib. Syst.*, 10(8):795–812, 1999.