Combining Batch Execution and Leasing Using Virtual Machines

Borja Sotomayor
University of Chicago
Chicago, IL, USA
borja@cs.uchicago.edu

Kate Keahey
Argonne National Laboratory
University of Chicago
Chicago, IL, USA
keahey@mcs.anl.gov

Ian Foster
Argonne National Laboratory
University of Chicago
Chicago, IL, USA
foster@mcs.anl.gov

ABSTRACT

As cluster computers are used for a wider range of applications, we encounter the need to deliver resources at particular times, to meet particular deadlines, and/or at the same time as other resources are provided elsewhere. To address such requirements, we describe a scheduling approach in which users request *resource leases*, where leases can request either as-soon-as-possible ("best-effort") or reservation start times. We present the design of a lease management architecture, Haizea, that implements leases as virtual machines (VMs), leveraging their ability to suspend, migrate, and resume computations and to provide leased resources with customized application environments. We discuss methods to minimize the overhead introduced by having to deploy VM images before the start of a lease. We also present the results of simulation studies that compare alternative approaches. Using workloads with various mixes of best-effort and advance reservation requests, we compare the performance of our VM-based approach with that of non-VM-based schedulers. We find that a VM-based approach can provide better performance (measured in terms of both total execution time and average delay incurred by best-effort requests) than a scheduler that does not support task pre-emption, and only slightly worse performance than a scheduler that does support task pre-emption. We also compare the impact of different VM image popularity distributions and VM image caching strategies on performance. These results emphasize the importance of VM image caching for the workloads studied and quantify the sensitivity of scheduling performance to VM image popularity distribution.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design—*Distributed systems*; D.4.5 [**Operating Systems**]: Reliability—*Checkpoint/restart*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Design, Management, Performance

1. INTRODUCTION

Many approaches have been developed to address the problem of providing computational resources to users. For example, an engineer wanting to run a simulation code may submit it as a batch job to a local or (via grid interfaces) remote cluster. A firm needing a web server for months or years may lease a dedicated server in a data center. A college instructor needing a small dedicated cluster for a few hours each week can obtain one from Amazon EC2 [37]. Each of these solutions is specialized to a specific usage scenario and only partially supports other usage patterns (if at all).

In our research, we seek to develop a resource provisioning model and system that can support many such usage scenarios at the same time. Motivated by this general goal, we focus in this paper on the specific problem of supporting workloads that combine requests for resources during a specific time period ("advance reservation" requests) and requests for resources whenever available ("best-effort" requests). The need for advance reservations arises, for example, when applications require coscheduling of multiple resources [38, 5], in urgent computing applications [25], in applications in which resource availability must coincide with some other event, such as a class [11, 38], and in applications expressible as a workflow of independent tasks that can be executed more efficiently by multilevel scheduling methods [29, 36, 16]. The need for best-effort resource allocation arises in many contexts and is supported by batch schedulers.

Although batch schedulers can and do include support for advance reservations, this mechanism often leads to utilization problems [11, 30, 31, 22], caused by the need to drain best-effort jobs from resources before a reservation can begin. Although checkpointing-based resource preemption can be used to minimize this effect, it requires either a checkpointing-capable OS (system-level checkpointing) or linking applications against checkpointing libraries (user-level checkpointing), both of which require specialized software that may not be available on all sites.

We propose here an approach that uses *leasing*, and not jobs, as the fundamental resource provisioning abstraction for both best-effort and advance reservation requests. The job abstraction used by batch schedulers ties together the provisioning of resources for the job and its execution, with resource provisioning typically happening as a side-effect of job submission. This forces resource consumers to use the provisioned resources through the job execution interfaces provided by a batch scheduler, instead of accessing the raw resources directly (e.g., in the simplest case, by logging into the resources). As a resource-provisioning mechanism, the job abstraction is insufficient for applications that are not easily expressible as jobs or where direct access to resources is required. A lease, on the other hand, is used only to provision resources, which can then be used at the user's discretion, including running jobs.

We develop lease terms for leases with best-effort availability as well as stricter availability constraints for advance reservation leases. Furthermore, using virtualization-based resource managers, we demonstrate that both can coexist without the utilization problems of advance reservations, without requiring applications to be modified (as required by user-level checkpointing) or needing to use a specific operating system (as required by system-level checkpointing). Hence, resource providers can satisfy the needs of batch computations, currently prevalent in scientific computing, while at the same time accommodating other usage scenarios. Our approach is based on using virtualization to suspend, migrate, and resume computations in virtual machines (VMs). In addition to leveraging these features of the VMs, we are in this way also able to provide leases with customized application environments.

Scheduling of best-effort and advance reservation jobs has been extensively studied [21, 23, 9, 11, 30, 31], but always within the context of batch job schedulers, and not resource leasing. Irwin et al. [17] (Shirako), Adabala et al. [1] (In-VIGO), Emeneker and Stanzione [7] (Dynamic Virtual Clustering), Fallenbeck et al. [8] (XGE), Kiyanclar et al. [19] (Maestro-VC), Nishimura et al. [24] and Yamasaki et al. [35] (Virtual Clusters-on-the-Fly), and Ruth et al. [28, 27] (VIOLIN/VioCluster) have researched VMs as a mechanism for resource management but without exploring the implications of running workloads that combine best-effort and advance reservation requests. Our previous work defines the virtual workspace abstraction to represent execution environments that are dynamically and securely deployed on remote resources through interoperable interfaces [18, 12] and an accurate resource model for VM-based resource management [33, 32].

In summary, our paper makes the following contributions:

- We describe a leasing-based architecture that integrates support for both best-effort and advance reservation leases.
- We describe an implementation of the lease-based architecture integrating those types of leases and explain how it will work within the current batch scheduling model.
- We show experimentally that our VM-based resource manager can provide resources more efficiently in certain cases, as measured by several resource utilization metrics.

2. RESOURCE LEASES

We define a lease as a negotiated and renegotiable agreement between a resource provider and a resource consumer, where the former agrees to make a set of resources available to the latter, based on a set of lease terms presented by the resource consumer. In this work we use "agreement" and "agreement terms" as defined by the WS-Agreement specification [2].

The lease terms encompass the *hardware resources* required by the resource consumer, such as CPUs, memory, and network bandwidth; a *software environment* required on the leased resources; and an *availability period* during which a user requests that the hardware and software resources be available. In previous work [13, 33], we focused on lease terms for hardware resources and a software environment. Our focus here is on the availability dimension of a lease. We consider here the following terms.

- *Start time* may be unspecified (a best-effort lease) or specified (an advance reservation lease). In the latter case, the user may specify either a specific start time or a time period during which the lease start may occur.
- *Maximum duration* refers to the total maximum amount of time that the leased resources will be available.
- Leases can be *preemptable*. A preemptable lease can be safely paused without disrupting the computation that takes place inside the lease.

We have developed an XML Schema for the above terms, extending the schema for resource allocations presented by Freeman et al. [13], but do not include it here because of space constraints. The XML representation of these terms could be used in a web service such as the Virtual Workspace Service [18] to negotiate a lease. In particular, a resource consumer would present the desired lease terms to the service, which would determine whether to accept or reject the request, and publish the accepted lease terms (which may be concretized by the service if the lease is accepted).

In this paper, we assume for simplicity that advance reservation leases are nonpreemptable and best-effort leases are preemptable. Furthermore, since we focus on availability, we make the simplifying assumptions that leases request a number of compute nodes with the same hardware requirements in each node and that the software environment is encapsulated inside a disk image (which could be used to reimage a hard drive or be used as a VM image). We further assume that, when determining whether to preempt a lease, a resource owner takes into consideration only the lease's preemptability (i.e., no other factors, such as priorities, would result in a decision not to preempt).

3. DESIGN AND IMPLEMENTATION

This section describes *Haizea*, a lease management architecture that enables resource consumers to negotiate the two kinds of leases described in the previous section. This architecture is composed of several components. Leases are requested through an *interface* component by using the XML language described in Section 2. These requests are then passed to a *scheduler* component. An enactment component is responsible for sending commands to nodes to start/end VMs, suspend/resume VMs, and initiate transfers. In this section we first describe the resource model for this architecture. Then we describe how leases are scheduled. Since a batch job scheduler could be a resource consumer (requesting leases to run best-effort jobs or advance reservation jobs), we discuss how our architecture can be used in conjunction with a job scheduler.

3.1 Resource Model

We assume that the lease management architecture manages W identical nodes each with a Virtual Machine Monitor (VMM) allowing the execution of virtual machines. Each node has P CPUs, M megabytes (MB) of memory, D MB of local disk storage, and a disk read/write transfer rate of H_r and H_w MB/s. We divide the disk space of each node into cache space D_p (used for caching disk images required by the VMs) and working space D_w (used for storing disk images of active and paused VMs), such that $D = D_p + D_w$. We assume that all required disk images are available in a *repository* from which they can be transferred to nodes as needed. For simplicity, we assume that the repository and nodes have the same characteristics and that all are connected at a bandwidth of B MB/s by a switched network with a nonblocking switch (i.e., the maximum transfer rate of the switch can support all the transfers).

A lease is implemented as a set of n VMs, each allocated resources described by a tuple (p, m, d, b), where p is number of CPUs, m is memory in MB, d is disk space in MB, and b is network bandwidth in MB/s. A disk image I with a size of s_I MB must be transferred from the repository to a node's cache space before the VM can start. When transferring a disk image to multiple nodes, we use multicasting and model the transfer time as $\frac{s_I}{B}$. Once a VM disk image is transferred to a node's cache space, one or more *disk image instances* (tied to a specific virtual machine) of the same size can be created by locally copying the cached disk image to the node's working space. We assume that this latter copy operation is performed gradually, using copy-on-write, and incurs no additional cost. We also assume that once a VM is terminated its disk image instance can be discarded.

If a lease is preempted, it is suspended by suspending its VMs, which may then be either resumed on the same node or migrated to another node and resumed there. Suspension results in the creation of a memory state file (the contents of the VM's memory) on the node where the VM is running, and resumption requires reading that image back into memory and then discarding the file. The size of the memory state file is m, and the time to suspend and resume a VM is $\frac{m}{H_w}$ and $\frac{m}{H_r}$ seconds, respectively. When a suspended VM is migrated to a different node, its disk image instance and memory state file are transferred, requiring $\frac{s_I+m}{B}$ seconds. Suspending multiple VMs communicating with each other can potentially disrupt communication between the nodes (e.g., messages that are lost "in flight" when the VMs are suspending). Emeneker et al. [6] showed that this problem can be avoided by suspending and resuming all VMs simultaneously, using NTP to synchronize these events, resulting in "a coherent network state with no timeouts." Since there are no TCP timeouts to recover from, and all suspend/resume operations must happen simultaneously, we assume that suspending and resuming multiple VMs also requires $\frac{m}{H_w}$ and $\frac{m}{H_r}$ seconds, respectively.

We assume in this paper that $P = 1$, that only one VM can be run per node, and that there is no contention between network traffic generated by applications running within VMs and network traffic associated with image movement—either because the applications running inside the VMs do not generate substantial network traffic or because there is a separate network for application network traffic. In future work, we will investigate the implications of relaxing these assumptions.

3.2 Lease Scheduling

We leverage our ability to suspend, migrate, and resume VMs to implement policies that seek to maximize scheduling performance by preempting certain (best-effort, preemptible) leases to make room for other (advance reservation, nonpreemptible) leases. In this way, we can use resources more efficiently than approaches that depend on node draining. In addition, we manage the transfer of disk images to ensure that required images are available on nodes when an advance reservation lease is scheduled; integrate disk image transfer into the scheduling process; and avoid image transfer operations, whenever possible, by managing caches of images on nodes. This model of explicitly scheduling VM overhead, instead of deducting it from the lease's execution time, was introduced in previous work [33, 32]. In effect, our scheduler schedules each lease as a workflow of actions such as image transfer, deployment, suspend/resume, and migration.

A lease is represented internally in our architecture by a *lease descriptor* that includes all the lease's terms (using the XML representation described in Section 2), along with zero or more *resource reservations*, representing the workflow of actions required by that lease. The scheduler keeps track of the available resources by using a slot table: when an action gets scheduled the corresponding resource reservations are marked in the slot table. The table has three dimensions: resource usage (the resource tuple (p, m, d, b) described earlier), the number of nodes, and the duration. In addition to representations for all available nodes, the slot table includes the image repository, so that the bandwidth required for image transfers from the repository to nodes can be accounted for.

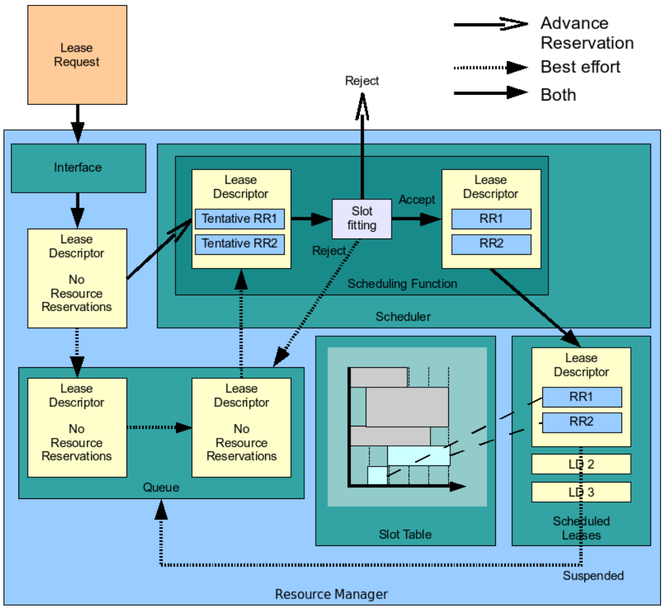

Figure 1: Submission and scheduling of a lease

When an image transfer is scheduled, the bandwidth in the origin and destination nodes is marked as completely used for the duration of the transfer, and the disk space in the destination node is decreased by d (if only the disk image is transferred) or by $d + m$ (if a VM is migrated). When VMs for a lease are scheduled on nodes, the resource usage tuple is decreased by the (p, m, d, b) amount of the requested resources on each node, for the duration of the VM. In the case of a preemptible lease, the working space on the disk is additionally decreased by m to hold the VM's memory in case it is suspended.

As noted earlier, a disk image required by a VM that is to be executed on a node is first transferred to the node's image cache. Images in a node's cache are reference counted based on the leases that depend on them. An image's reference count is decremented each time a lease that depends on this image ends. If the image pool is full when an image is added, the scheduler removes the least recently used image with reference count equal to zero.

Figure 1 summarizes how lease descriptors make their way through the lease manager, from submission to completion. Incoming advance reservation leases are always scheduled right away, whereas best-effort leases are put on a queue. Each scheduled lease has a pointer to the resources it is assigned in the slot table, so that those resources can be released at the end of the lease.

Each incoming best-effort lease is placed at the end of the queue. The scheduling function periodically evaluates the queue, using an aggressive backfilling algorithm [21, 23], to determine whether any best-effort leases can be scheduled. The scheduler first checks, for each node, the time when the required disk image can be made available on the node (by using a cached image, by piggybacking on a scheduled multicast, or by scheduling a separate transfer), and the time the node's resources will be available and takes the later of the two. These potential start times are sorted, and the n nodes with the earliest times are selected, giving preference first to nodes where the lease can run without having to be suspended; the latest time in this group is the time for which the lease will be scheduled. The scheduling of the lease also results in the scheduling of the dependency actions, that is, image transfers (if needed) or incrementing of reference counts of a cached image.

Note that a lease may be scheduled even if there is a "blocking" lease, such as an advance reservation lease scheduled in the future, that would prevent a best-effort lease to run for its entire requested duration before the blocking lease starts. In such a case, the VMs in the lease may be suspended before a blocking lease. The remainder of a suspended lease is put back in the queue, according to its submission time, and scheduled as if it were another best-effort request. If the scheduler maps the remaining duration to nodes different from those where the lease was suspended, the VM image and memory state file will be transferred to the new nodes before the lease can resume.

When an advance reservation lease is requested, the scheduling function first determines whether the request is viable, that is, whether sufficient resources are available at the requested time and whether necessary image transfers can be scheduled as needed. If the lease is found not to be viable, it is rejected. Other than this, we assume an "accept all" policy, such that advance reservation leases are always accepted as long as there are sufficient resources available for them. If the lease is accepted, the scheduler determines what nodes can support the lease. The scheduler first chooses nodes that will not require preempting another lease and then chooses nodes for which the required image can be found in the node's image cache (we found that reversing this policy had no significant effect in the context of our workloads). The exception to this policy is when a lease will be viable only if images are reused, in which case priority is given to nodes with a reusable copy of the required image.

For advance reservation leases, image transfers are scheduled using an earliest deadline first (EDF) algorithm [26], where the deadline for the image transfer is the start time of the lease. Since the start time of an advance reservation lease may occur long after the lease request, we modify the basic EDF algorithm so that transfers take place as close as possible to the deadline, preventing images from unnecessarily consuming disk space before the lease starts.

3.3 Combining Job and Lease Management

Our architecture is designed so that it can potentially act as a resource provisioning backend for existing job management architectures (e.g., as a scheduler for Torque or as a decision-making module for SGE's schedd), allowing users to continue to submit their computations in the form of jobs. When a job is submitted, the execution manager must request a best-effort lease for the job. Once the lease is accepted, the lease manager and the execution manager communicate through a series of events to coordinate their actions. The following events are sent from the lease manager to the execution manager:

Lease ready: Once a lease request is accepted, the lease manager must still provision resources on a best-effort basis, which may require queuing the lease request until it can be satisfied. When resources become available, the lease manager informs the execution manager of the resources that have been allocated to the job.

Lease suspended/resumed: If a best-effort lease must be preempted, the lease manager notifies the execution manager that the resources where the job is running will become unresponsive, so that it will not assume the job has failed or the nodes have crashed. Another event is sent when the lease is resumed.

The execution manager, on the other hand, will send a "Release lease" event when a job concludes its execution or is cancelled, and no longer requires the leased resources.

4. EXPERIMENTAL RESULTS

We used a discrete event simulator to evaluate the performance of our lease management architecture for various workloads. We present here our simulation model, the workloads we used, and the results of our experiments.

4.1 Simulation Model

We perform a discrete event simulation of the submission, scheduling, and execution of best-effort and advance reservation leases. The input to the simulator is (a) description of the simulated cluster and (b) a trace: a sequence of best-effort and advance reservation lease requests, each specified in the format described in Section 2 and annotated with the time at which the request was submitted. The submission of a lease results in an event that causes the scheduler adding entries to the slot table (if the lease is accepted). The start and the end of an allocation in the slot table is also treated as an event that causes the scheduler to reevaluate the schedule.

4.2 Workloads

We construct the workloads used in our experiments by adapting the SDSC Blue Horizon cluster job submission trace from the Parallel Workloads Archive [39]. In general terms, we take a set of job submission requests from that trace and treat them as a set of best-effort lease requests and then insert an additional set of advance reservation requests. Keeping the best-effort requests fixed, we vary the advance reservation requests to obtain a set of 72 different workloads.

More specifically, we use as our best-effort requests the first 30 days of requests in the SDSC Blue Horizon trace, For each of these 5,545 requests, we extract from the trace its submission time, requested duration, and requested number of nodes. (In this 30-day extract, 66.85% of the requests have a requested duration of one hour or less, and 64.09% of the requests require four nodes or less.) We also set the per-node resource allocation to $p = 1$ and $m = 1024$. For simulation purposes, we take from the trace the actual run time, which is usually less than the requested job duration. Thus, in our simulations, best-effort leases frequently complete "prematurely."

To generate our workloads, we then interleave with this set of best-effort requests a set of advance reservation requests, generated according to three parameters:

- ρ, the aggregate duration of all advance reservation leases in a trace, computed as a percentage of the total CPU hours in the simulation's run, which is the number of nodes multiplied by the time when the last best-effort request is submitted. We use the values ρ =5%, 10%, 15%, 20%, 25%, and 30%. (We do not explore larger values because the trace's utilization is 76.2%, according to the Parallel Workloads Archive.)
- δ, the duration of each advance reservation lease, for which we use average values of 1, 2, 3, or 4 hours. (The duration is selected randomly from a range spanning $\delta \pm$ 30m.)
- ν, the number of nodes requested by each lease, for which we use three ranges, from which the value is selected using a uniform distribution: *small* (between 1 and 24), *medium* (between 25 and 48), or *large* (between 49 and 72).

Our use of ρ allows us to compare results obtained with traces with different duration (δ) and size (ν) parameters, by ensuring that all traces with the same ρ value involve the same amount of extra work.

Given values for ρ, δ, and ν, we then determine the arrival times of the advance reservation requests as follows. First, we determine the number of requests that will be generated, and we divide that number into 30 days to obtain an average interlease interval i. Then, we choose the intervals between requests at random in the range ($i - 1$ hour, $i + 1$ hour). Thus, the smaller the average lease duration, the more frequent is the arrival of requests (since there will be more advance reservation lease requests). Similarly, the smaller the average number of nodes, the higher the frequency. We further constrain advance reservation lease requests to involve an advance notice of exactly H hours. In the experiments described below, we set $H = 24$, unless otherwise noted. As with the best-effort requests, the advance reservation lease requests have a per-node resource allocation of $p = 1$ and $m = 1024$.

In our experiments, we explore every combination of the parameters ρ, δ, and ν, for a total of 72 workloads. We refer to workloads using the notation [ρ%/δH/ν] (e.g., [10%/2H/medium]).

We perform experiments using three criteria for disk image selection:

single: assign the same disk image to each request.

uniform: assign one of 37 possible images to each requests at random using a uniform distribution.

skewed: assign one of 37 possible images using a distribution where 7 images each have a 10% probability of being selected, and the remaining 30 images each have a 1% probability of being selected.

All images have a disk size of 4GB.

4.3 Simulated Cluster

Our simulated cluster is modeled after the SDSC Blue Horizon cluster. It comprises 144 single-CPU nodes, each with 40 GB of disk (with $D_p = 20$ and $D_w = 20$) and 1 GB of memory, connected with a switched Ethernet network (100 Mb/s, we conservatively assume a 10 MB/s bandwidth). For the purposes of estimating the time required to suspend and resume a VM, we assume H_w and H_r to be 50 MB/s, based on results presented by Fallenbeck et al. [8]. We conservatively assume that the sum of the boot-up and shutdown time of a VM does not exceed 20 seconds. To account for the slowdown produced by running inside a VM, we assume that any computation running inside a VM requires 5% more time to run. Further, we assume that the time required to send commands from the resource manager to the nodes is negligible and that the hardware will not behave erratically, and we inject no hardware failures into the simulated cluster. We plan to relax the no-failure assumption in future work.

4.4 Experiments

We ran our simulator using each of the workloads described above, with the following configurations:

NOVM–NOSR — No VM, no suspend/resume: Leases do not run on VMs. We follow the resource model of Section 3, except there is no VM image to transfer and no 5% runtime slowdown. In addition, leases cannot be suspended or resumed; thus, a preempted lease is cancelled and requeued.

NOVM–SR — No VM, with suspend/resume and migration: Like NOVM–NOSR, but leases can be suspended and resumed. This configuration represents a job scheduler capable of checkpointing and migrating any job. We assume that, to be able to checkpoint any application, system-level checkpointing is used, and that suspending a lease requires saving the entire memory to disk, as would happen in the VM cases.

VM–PREDEPLOY — With VM, single predeployed image: The leases run on VMs, following the resource model described in Section 3, but all leases use the same VM image, which is assumed to be predeployed on the nodes.

VM–MULT — With VM, multiple images: As VM–PREDEPLOY, but using the *uniform* list of requests and removing the assumption that images are predeployed (i.e., an image transfer has to be scheduled before the VM can start). Images are not reused on the nodes.

VM–REUSE–UNIFORM and VM–REUSE–SKEWED — With VM, multiple images with image reuse: Same as VM–MULT, but reusing images on the nodes. We use the *uniform* and *skewed* lists of requests.

During the experiments, we observe, for each lease, the times t_a (the arrival time, or time when the lease request is submitted), t_s (the start time of the lease) and t_e (the time the lease ends). At the end of an experiment, we compute the following metrics:

all-best-effort: We define all-best-effort as the time from the start of the trace to when the last best-effort request is completed. We normalize this value by presenting the relative difference between this time and the time required to run all the best-effort requests *without* advance reservation leases in configuration NOVM–NOSR (this time is 2,674,265 seconds, or roughly 30.95 days). Thus, a value x indicates that an experiment took $2,674,265 \cdot x$ to run (with $x = 1.0$ meaning that the experiment took the same time as the baseline case).

Wait time of best-effort requests: We define wait time as $t_s - t_a$, the time a best-effort request must wait before it starts running.

Bounded slowdown of best-effort requests [10]: If t_u is the time the lease would take to run on a dedicated physical system (i.e., not in a VM), the lease's slowdown is $\frac{t_e - t_a}{t_u}$. If t_u is less than 10 seconds, the bounded slowdown is computed the same way, but assuming t_u to be 10 seconds [10].

When computing the last two metrics, we discard the first 5% of measurements, to avoid ramp-up effects. We retain the ramp-down period because this is where we observe the wait times and slowdowns of the requests that languish in the queue until there are no more advance reservation leases.

Figure 2 shows all-best-effort results for all experiments. This metric provides a good measure of utilization, taking into account both VM runtime slowdown and the overhead of VM deployment. We observe that using suspend/resume and migration (with and without VMs) results in a shorter run time than NOVM–NOSR in every case. In fact, using suspend/resume and migration produces a slowdown of, at most, 10% relative to not injecting any advance reservation leases at all, whereas not using suspend/resume can produce a slowdown of up to 32.97%. Additionally, the duration of the advance reservation leases is more likely to affect this metric in the NOVM–NOSR configuration, with shorter-duration (and thus shorter-interval) leases producing the worst results.

As mentioned previously, all advance reservation leases were submitted to the scheduler with an advance notice of 24h. In preliminary tests with fewer workloads, based on a 15-day trace, we investigated the effect of using different advance notices (1h, 3h, 6h, 12h, and 24h). We observed that the advance notice can have an effect when suspend/resume is not used, with all-best-effort tending to increase for shorter advance notices (e.g., in the most extreme

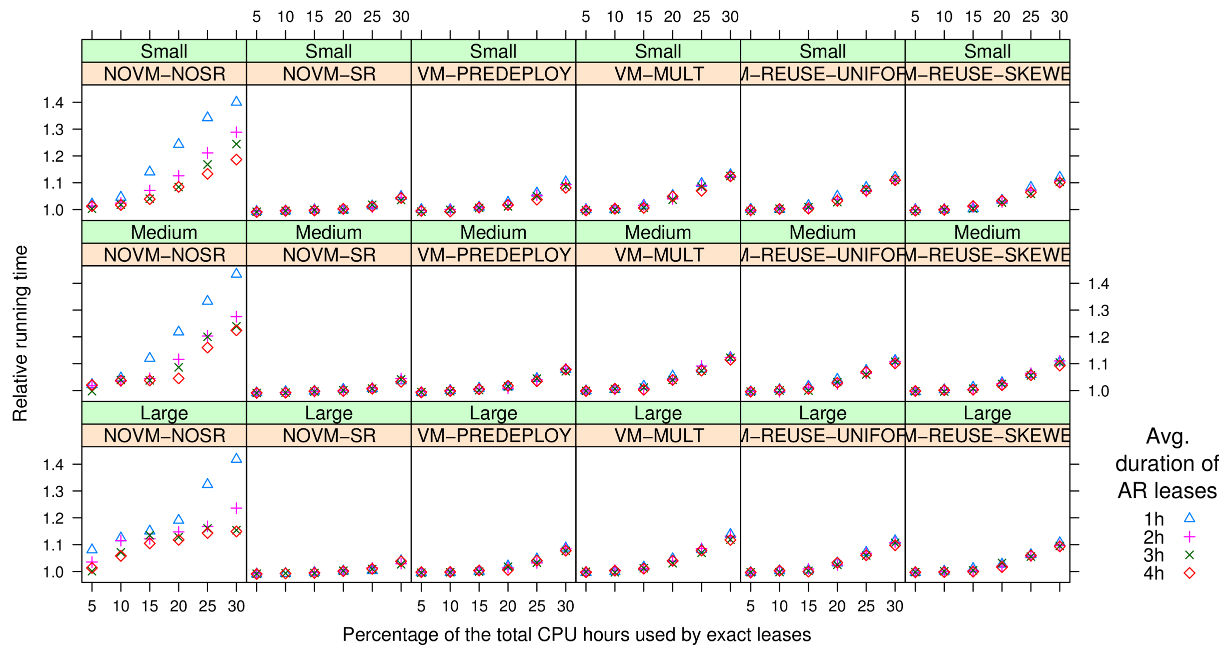

Figure 2: Each data point in this graph represents the all-best-effort metric in each experiment. The x axis represents ρ and the y axis represents the value of all-best-effort (normalized as described in the text). The graphs are grouped by ν (the number of nodes requested by each lease) and the experiment configuration. The symbol of each point denotes the value of δ (see legend).

case, observed with workload [10%/2H/small] using a 1h advance notice, all-best-effort increases by 46.69% compared to using a 24h notice). We attribute this effect to the increasing likelihood that an advance reservation lease will preempt a best-effort lease that has already been scheduled, necessitating that it be cancelled and resubmitted. When using suspend/resume, on the other hand, the effect is negligible.

While all-best-effort gives a good measure of effective utilization (indicating how much faster an entire workload will be run from beginning to end), it does not say much about individual leases, which requires looking at the other two metrics. For example, an inspection of the run history reveals that the longer total run time when not using suspend/resume (in NOVM–NOSR) is due to best-effort leases that remain in the queue and are not run until the ramp down period, when no more best-effort leases are being submitted. These leases remain in the queue because the advance reservation leases prevent them from being scheduled, especially when the interval between advance reservation leases is short. Because of space limitations, we are unable to present the data for wait time and slowdown of best-effort requests from all the cases; instead, we constrain most of our discussion to the cases [10%/3H/medium], [20%/3H/medium], and [30%/3H/medium], which are representative of the trends that we observe across all cases.

Table 1 shows the total run time, average wait time, and average slowdown for these three cases. Although the run time metric is always the worst (longest) in the NOVM–NOSR configuration, in the other two metrics the NOVM–NOSR configuration results in better performance (shorter average wait times and smaller average slowdowns) in most cases. We attribute this result to the fact that, without suspend/resume, the scheduler must rely heavily on backfilling to efficiently use the time and space before a blocking lease (such as an advance reservation lease). This behavior will favor short leases, which "skip the queue" when used as backfill. Since the majority of best-effort leases in our trace are shorter than one hour, they are ideal candidates for backfilling. Thus, many such leases end up with short wait times and small slowdowns. However, suspend/resume need not look ahead for shorter best-effort leases when backfilling: it can simply take the next best-effort lease, even if long, knowing that it can suspend this lease if it has not run to completion before a blocking lease is scheduled. Paradoxically, the aggregate effect of not preferentially selecting shorter leases for backfill can be to increase average wait time and slowdown.

We can support this observation by looking at how wait times and the slowdown vary with requested duration and number of nodes. Figure 3 shows the regression curves (using the Lowess smoother [4]) for these metrics and variables for [20%/3H/medium]. The left two graphs show how wait time and slowdown vary with the requested lease duration. We see that short leases in the NOVM–NOSR configuration have shorter wait times and smaller slowdowns than all other configurations (which use suspend/resume). However, the tendency is for both the wait time and slowdown to increase as the requested duration increases since, as noted above, backfilling favors short requests. When using suspend/resume, on the other hand, we observe that the trend is for the wait times to not vary with the requested duration and for the slowdowns to exhibit a slight decreasing trend. Thus, all requests are treated more fairly, although the overall average does increase (the shorter requests, which make up the majority, have longer wait times because they no longer "skip the queue," thanks to backfilling). We note that this is a desirable effect in many scenarios as it provides a more "democratic" treatment for different application types. Fallenbeck et al. [8] used similar strategies to equalize wait times for different application groups.

If we look only at the suspend/resume configurations, we can see that VM–MULT performs the worst, since no attempt is being made to reuse VM images on the nodes. Adding image reuse (VM–REUSE–UNIFORM and VM–REUSE–SKEWED) reduces wait times and slowdowns, although performance is still not as good

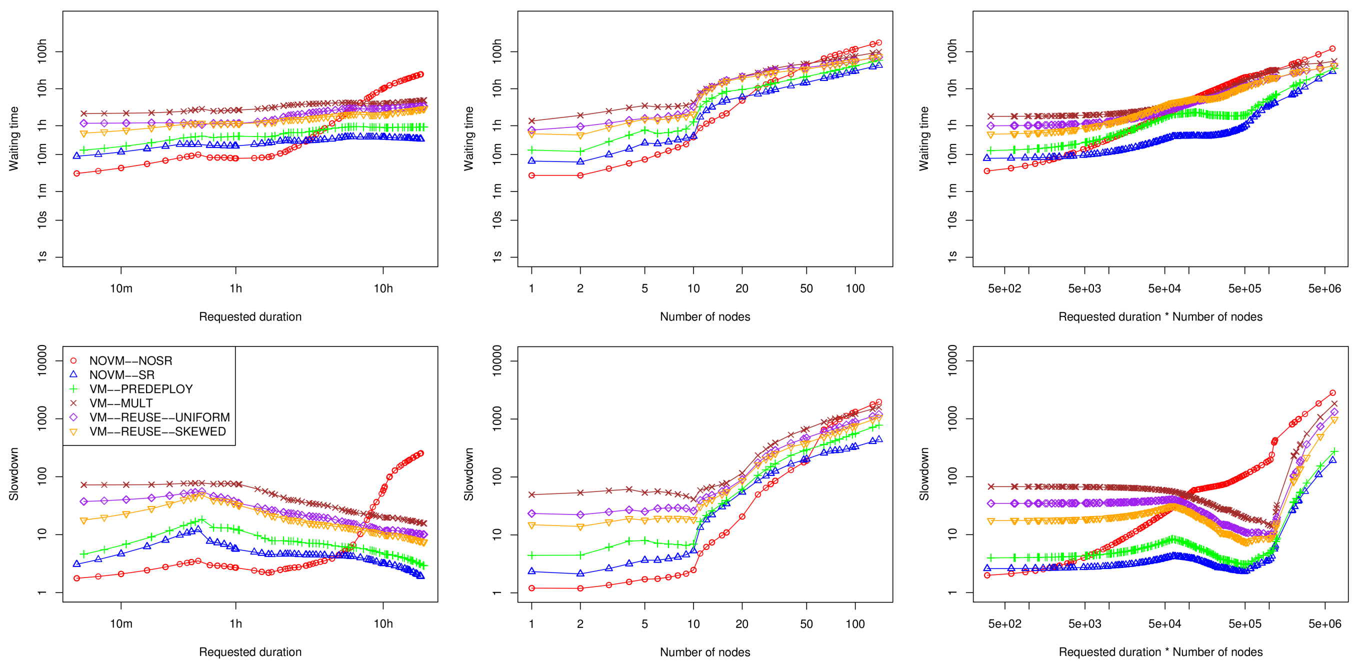

Figure 3: Lowess curves for [20%/3H/medium]

as when using a single predeployed image (VM–REUSE–PREDEPLOY).

The center two graphs show how the wait time and slowdown vary with the number of requested nodes. When looking just at the effect of node counts, we see that adding suspend/resume has little effect. In fact, we can observe the same trend across all configurations: wait time and slowdown do not vary when the node count is less than ten but tend to increase for larger node counts. If we observe how the wait time and slowdown vary with the requested CPU time (the product of the requested duration and the requested number of nodes) in the right two graphs, we see trends similar to those when looking just at the requested duration, except that the upward trend as the CPU time increases is more pronounced in the NOVM–NOSR configuration. This upward trend also is evident in the suspend/resume configurations, but only for large values of CPU time.

In general, we observe across all 72 workloads that, as ρ increases, wait times and slowdowns tend to increase more sharply in the NOVM–NOSR configurations, but tend not to vary in the configurations that use suspend/resume.

5. RELATED WORK

The problem of scheduling best-effort jobs has been extensively studied. The most commonly used optimization is backfill [21, 23, 9]. We leverage the suspend/resume capability of virtual machines to extend backfill algorithms, allowing leases to be suspended before the start of a reservation, whether an advance reservation lease or a parallel best-effort lease reserved by a backfilling algorithm. Jobs running on nonvirtualized machines can also be preempted and have their state saved to disk (*checkpointing*), to achieve more fault-tolerant software or to optimize job scheduling. However, OS-level checkpointing requires applications to be compiled for the checkpointing-aware OS, and may require relinking an application with new libraries (e.g., the BLCR kernel [15] can checkpoint MPI applications only if these are linked with BLCR-aware MPI libraries). The use of VMs allows any computation to be transparently checkpointed without modifying the application or porting it to a new architecture. Although nodes must use a specific VMM kernel, the guest VMs can potentially run any operating system.

Several modern batch schedulers, such as SGE and Maui, support advance reservations, which allow users to request strict availability periods, similar to the advance reservation leases described in this paper. Once a reservation is made, however, users must still interact with those resources using the job abstraction (typically through a queue that is created specifically to submit jobs to the reserved resources) and cannot request a custom software environment. Additionally, advance reservations can cause utilization problems in clusters [11, 30, 31, 22] as a result of the strict constraints they impose on the job schedule, which is only partially alleviated by backfilling. Nonetheless, there is a growing interest in supporting advance reservation capabilities in systems such as TeraGrid [38, 22], for a variety of usage cases, such as coscheduling of large parallel jobs across sites. Our work facilitates support of the leasing semantics required by advance reservations, while having a smaller impact on utilization and queue wait times than do existing approaches.

Fallenbeck et al. [8] extended the SGE scheduler to use the save/restore functionality of Xen VMs, allowing large parallel jobs to start earlier by suspending VMs running serial jobs, and resuming them after the large parallel job finished. Emeneker et al. [7] extended the Moab scheduler to support running jobs inside VMs, and explored different caching strategies for faster VM image deployment on a cluster. However, both studies use VMs only to support the execution of best-effort jobs and do not currently schedule image transfers separately; moreover, the Moab work does not integrate caching information into scheduler decisions.

The OAR batch scheduler, used in the Grid5000 [3] architecture, is coupled to a node reconfiguration system that can deploy a software environment, using disk images, on a node before a job starts running. However, these disk images are used to reimage physical

Table 1: Experiment running times, average waiting times, and average slowdowns for [10%/3H/medium], [20%/3H/medium], and [30%/3H/medium]

Time to complete best-effort leases, relative to time without advance reservation leases

$\rho \rightarrow$	10%	20%	30%
NOVM–NOSR	3.91%	8.66%	23.93%
NOVM–SR	-0.76%	0.10%	4.14%
VM–PREDEPLOY	-0.27%	1.58%	7.19%
VM–MULT	0.60%	3.95%	12.21%
VM–REUSE–UNIFORM	0.03%	3.15%	10.74%
VM–REUSE–SKEWED	-0.36%	2.48%	10.43%

Average waiting time for best-effort leases, in thousands of seconds

$\rho \rightarrow$	10%	20%	30%
NOVM–NOSR	16.91	23.91	68.95
NOVM–SR	6.18	8.14	14.84
VM–PREDEPLOY	8.05	12.17	22.89
VM–MULT	15.69	34.56	78.83
VM–REUSE–UNIFORM	12.31	25.68	62.52
VM–REUSE–SKEWED	10.69	22.85	58.44

Average bounded slowdown

$\rho \rightarrow$	10%	20%	30%
NOVM–NOSR	60.48	71.80	296.74
NOVM–SR	31.05	49.56	134.91
VM–PREDEPLOY	39.60	62.45	133.59
VM–MULT	91.94	203.39	441.10
VM–REUSE–UNIFORM	62.24	140.42	338.60
VM–REUSE–SKEWED	50.46	115.99	309.67

drives, instead of being used by VMs. Although this approach allows practically any environment to be deployed, it increases the deployment overhead which, in OAR, is not scheduled separately from the job.

Several groups have investigated the use of *multilevel scheduling* at the cluster level by using existing job schedulers to provision compute resources, which are then managed by a separate scheduler, instead of being limited to using the job execution semantics provided by the job scheduler that provisioned the resources. The Condor scheduler's glidein mechanism [14] was the first to apply this model on compute clusters, by starting (or "gliding in") Condor daemons on the provisioned resources. The MyCluster project [34] similarly allows Condor or SGE clusters to be overlaid on top of TeraGrid resources to provide users with personal clusters. The Falkon task scheduler [16] can also be deployed through a GRAM interface on compute resources and is optimized to manage the execution of lightweight tasks. In all these approaches, resource provisioning is not completely decoupled from the job scheduler, and there is no way to obtain a custom software environment.

Several datacenter-based solutions have emerged that completely decouple resource provisioning from job submission, including server hosting providers that provide long-term leases over virtualized resources. More recently, Amazon's EC2 [37] introduced the notion of *cloud computing*, where virtual machines are dynamically provisioned immediately with customized software environments and use is charged by the hour. These solutions excel at providing users with exactly the software environment required, and most provide a large number of hardware options; however, they support few types of availability periods. Server hosting providers are geared toward long availability periods, while EC2 requires resources to be provisioned immediately.

Several groups have explored the use of VMs to create "virtual clusters" on top of existing infrastructure. Nishimura et al.'s [24] system for rapid deployment of virtual clusters can deploy a 190-node cluster in 40 seconds. Their system accomplishes this impressive performance by representing software environments as binary packages that are installed on the fly on generic VM images. They optimize installation by caching packages on the nodes, thus reducing the number of transfers from a package repository. This approach limits the possible software environments to those that are expressible as installable binary packages (which is not always possible) but does provide a faster alternative to VM image deployment if the installation time is short enough. Yamasaki et al. [35] improved this system by developing a model for predicting the time to completely set up a new software environment on a node, allowing their scheduler to choose nodes that minimize the time to set up a new virtual cluster. Whereas our model assumes homogeneous nodes and uses disk images to encapsulate software environments (and thus can use a simple formula for estimating the time to set up a software environment), Yamasaki et al.'s model takes node heterogeneity into account and uses the parameters of each node (CPU frequency and disk read/write speeds) and empirical coefficients to predict the time to transfer and install all required packages, and then reboot the node. However, their model does not include an availability dimension and assumes that all resources are required immediately (a subset of advance reservation leases), while our model allows for deployment overhead to be scheduled in different ways depending on the lease's requested availability (best-effort or advance reservation).

The Shirako system [17] uses VMs to partition a physical cluster into several virtual clusters. This work also relies on the lease abstraction, allowing users to obtain resource leases that can be redeemed in the future. Their model assumes that any overhead involved in deploying and managing the VM will be deducted from the lease's availability. In contrast, we seek to provide a guaranteed set of resources during an agreed-upon availability period, adequately managing overhead in such a way that this guarantee is not breached. Previous work [33, 32] showed that not managing this overhead can prevent advance reservation leases from starting at their specified start time and thus would prevent a resource provider from entering into agreements that guarantee the lease's availability period.

Other groups have explored a variety of challenges involved in deploying and running a virtual cluster, including virtual networking and load balancing between multiple physical clusters (VIOLIN/VioCluster [28, 27]), automatic configuration and creation of VMs (In-VIGO [1] and VMPlants [20]), and communication between a virtual cluster scheduler and a local scheduler running inside a virtual cluster (Maestro-VC's two-level scheduling [19]). However, they do not explore workloads that combine best-effort and advance reservation requests, nor do they schedule deployment overhead of VMs separately.

6. FUTURE WORK

Our architecture makes certain simplifying assumptions on several policy points. Most notably, we (a) accept all lease requests, except advance reservation leases for which no resources can be provisioned (e.g., because an advance reservation lease has already been scheduled on those resources) and (b) assume that best-effort requests are always preemptible and advance reservation leases are not. In future work, we will explore policies that accept or reject leases based on criteria other than resource availability and the effect of different policies on performance. For example, we have only scratched the surface of the effect that advance notice (of advance reservation leases) can have on global performance. If a system administrator allows for short advance notices, users will presumably prefer to request advance reservation leases instead of having their requests wait in a queue, increasing the advance reservation lease workload, which this work has shown can result in worse performance. On the other hand, requiring long advance notices will reduce the value of advance reservations to users. Future work will investigate the effect that these policy points have on performance and ways to configure them in order to balance the requirements and needs of both resource providers and users. We will also look at preemptible advance reservation leases and nonpreemptible best-effort leases and will investigate policies that take into account additional factors (such as priorities) when deciding when a lease should, or should not, be preempted.

We also plan to develop a prototype implementation of our architecture that can manage real resources. An important issue will be handling hardware failures. Our experiments currently assume no hardware failures. Our scheduler could easily deal with failures by cancelling and resubmitting any best-effort leases affected by a failure. However, this strategy can be inefficient if, for example, a long lease is cancelled when most of its work is done. We can minimize the impact of failures by performing periodic checkpoints (using VM checkpointing mechanisms), at the cost of extra disk space. Another concern is that cancelling an advance reservation lease breaches the agreement made with the resource consumer. We will explore strategies that make advance reservation leases more resilient, such as overreserving resources in anticipation of failures.

We will also explore other types of leases, such as deadline-sensitive best-effort leases, and more complex lease terms, such as availability periods divided into multiple segments, each with different hardware resources (e.g., a lease may need network bandwidth during the first 10 minutes of an application's execution, to allow download information from a third site, but not require network connectivity for the remainder of the lease). Additionally, we will explore dynamic renegotiation of lease terms and lease metascheduling.

7. CONCLUSIONS

We have described a lease management architecture that allows resource consumers to request resource leases with semantics that encompass hardware resource, software environments, and availability. We have focused on resource availability and, in particular, on two types of availability that are frequently required in practice: *preemptible best-effort*, roughly corresponding to the availability requirements of batch computations, and *nonpreemptible advance reservation*. Resource leases are a general-purpose abstraction for provisioning resources. They do not limit the resource consumer to access and use those resources through constrained interfaces, such as those of a job scheduler, which requires users to specify a single computation to be performed on the requested resources.

We have presented experimental results that show that, when using workloads that combine best-effort and advance reservation requests, a VM-based approach with suspend/resume can overcome the utilization problems typically associated with the use of advance reservations. Our results show that, even in the presence of the runtime overhead resulting from using VMs, a VM-based approach results in consistently better total execution time than a scheduler that does not support task preemption, and only slightly worse performance than a scheduler that does support task preemption. Measurements of wait time and slowdown for the same experiments show that, although the average values of these metrics increase when using VMs, this effect is due to short leases not being preferentially selected as backfill before a blocking lease. In effect, a VM-based approach does not favor leases of a particular length over others, unlike systems that rely more heavily on backfilling. Moreover, we show that, although supporting the deployment of multiple software environments, in the form of multiple VM images, requires the transfer of potentially large files, this deployment overhead can be minimized through the use of image transfer scheduling and caching strategies.

8. ACKNOWLEDGMENTS

This work was supported by NSF CSR award #527448 and in part, by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, SciDAC Program, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

We thank Dan Nurmi for his suggestions on how to analyze our data, and Rich Wolski and our anonymous reviewers for their detailed and insightful comments.

9. REFERENCES

[1] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu. From virtualized resources to virtual computing grids: the In-VIGO system. *Future Gener. Comput. Syst.*, 21(6):896–909, June 2005.

[2] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web services agreement specification (WS-Agreement).

[3] R. Bolze, F. Cappello, E. Caron, M. Daydé , F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and T. Irena. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, Nov. 2006.

[4] W. S. Cleveland. Lowess: A program for smoothing scatterplots by robust locally weighted regression. *The American Statistician*, 35(54), 1981.

[5] K. Czajkowski, I. Foster, and C. Kesselman. Resource co-allocation in computational grids. In *HPDC '99: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, page 37, Washington, DC, USA, 1999. IEEE Computer Society.

[6] W. Emeneker and D. Stanzione. Increasing Reliability through Dynamic Virtual Clustering. In *High Availability and Performance Computing Workshop*, 2006.

[7] W. Emeneker and D. Stanzione. Efficient Virtual Machine Caching in Dynamic Virtual Clusters. In *SRMPDS Workshop, ICAPDS 2007 Conference*, December 2007.

[8] N. Fallenbeck, H.-J. Picht, M. Smith, and B. Freisleben. Xen and the art of cluster scheduling. In *VTDC '06: Proceedings of the 1st International Workshop on Virtualization Technology in Distributed Computing*, Washington, DC, USA, 2006. IEEE Computer Society.

[9] D. Feitelson, L. Rudolph, and U. Schwiegelshohn. Parallel job scheduling – a status report. *10th Workshop on Job Scheduling Strategies for Parallel Processing, New-York, NY.*, 2004.

[10] D. G. Feitelson and L. Rudolph. Metrics and benchmarking for parallel job scheduling. *Lecture Notes in Computer Science*, 1459:1+, 1998.

[11] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *Proceedings of the International Workshop on Quality of Service*, 1999.

[12] I. T. Foster, T. Freeman, K. Keahey, D. Scheftner, B. Sotomayor, and X. Zhang. Virtual clusters for grid communities. In *CCGRID*, pages 513–520. IEEE Computer Society, 2006.

[13] T. Freeman, K. Keahey, I. T. Foster, A. Rana, B. Sotomayor, and F. Wuerthwein. Division of labor: Tools for growing and scaling grids. In *ICSOC*, 2006.

[14] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.

[15] P. H. Hargrove and J. C. Duell. Berkeley lab checkpoint/restart (blcr) for linux clusters. *Journal of Physics: Conference Series*, 46:494–499, 2006.

[16] I.Raicu, Y.Zhao, C.Dumitrescu, I.Foster, and M.Wilde. Falkon: a fast and light-weight task execution framework. In *IEEE/ACM International Conference for High Performance Computing, Networking, Storage, and Analysis (SC07)*, 2007.

[17] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. G. Yocum. Sharing networked resources with brokered leases. In *USENIX Technical Conference*, June 2006.

[18] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life on the grid. *Scientific Programming*, 13(4):265–276, 2005.

[19] N. Kiyanclar, G. A. Koenig, and W. Yurcik. Maestro-VC: A paravirtualized execution environment for secure on-demand cluster computing. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, page 28, Washington, DC, USA, 2006. IEEE Computer Society.

[20] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo. Vmplants: Providing and managing virtual machine execution environments for grid computing. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 7, Washington, DC, USA, 2004. IEEE Computer Society.

[21] D. A. Lifka. The ANL/IBM SP scheduling system. In *IPPS '95: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 295–303, London, UK, 1995. Springer-Verlag.

[22] M. W. Margo, K. Yoshimoto, P. Kovatch, and P. Andrews. Impact of reservations on production job scheduling. In *13th Workshop on Job Scheduling Strategies for Parallel Processing*, 2007.

[23] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Trans. Parallel Distrib. Syst.*, 12(6):529–543, 2001.

[24] H. Nishimura, N. Maruyama, and S. Matsuoka. Virtual clusters on the fly - fast, scalable, and flexible installation. In *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 549–556, Washington, DC, USA, 2007. IEEE Computer Society.

[25] P.Beckman, S.Nadella, N.Trebon, and I.Beschastnikh. SPRUCE: A system for supporting urgent high-performance computing. *IFIP International Federation for Information Processing, Grid-Based Problem Solving Environments*, 239:295–311, 2007.

[26] K. Pruhs, J. Sgall, and E. Torng. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter Online Scheduling. CRC Press, Inc., Boca Raton, FL, USA, 2004.

[27] P. Ruth, P. McGachey, and D. Xu. VioCluster: Virtualization for dynamic computational domains. *Proceedings of the IEEE International Conference on Cluster Computing (Cluster'05)*, 2005.

[28] P. Ruth, J. Rhee, D. Xu, R. Kennell, and S. Goasguen. Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure. *IEEE International Conference on Autonomic Computing, 2006.*, 2006.

[29] G. Singh, C. Kesselman, and E. Deelman. Performance impact of resource provisioning on workflows. Technical Report 05-850, Department of Computer Science, University of South California, 2005.

[30] W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In *IPDPS '00: Proceedings of the 14th International Symposium on Parallel and Distributed Processing*, page 127, Washington, DC, USA, 2000. IEEE Computer Society.

[31] Q. Snell, M. J. Clement, D. B. Jackson, and C. Gregory. The performance impact of advance reservation meta-scheduling. In *IPDPS '00/JSSPP '00: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 137–153, London, UK, 2000. Springer-Verlag.

[32] B. Sotomayor. A resource management model for VM–based virtual workspaces. Master's thesis, University of Chicago, February 2007.

[33] B. Sotomayor, K. Keahey, and I. Foster. Overhead matters: A model for virtual resource management. In *VTDC '06: Proceedings of the 1st International Workshop on Virtualization Technology in Distributed Computing*, page 5, Washington, DC, USA, 2006. IEEE Computer Society.

[34] E. Walker, J. Gardner, V. Litvin, and E. Turner. Creating personal adaptive clusters for managing scientific tasks in a distributed computing environment. In *Challenges of Large Applications in Distributed Environments*, 2006.

[35] S. Yamasaki, N. Maruyama, and S. Matsuoka. Model-based resource selection for efficient virtual cluster deployment. In *VTDC '07: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, 2007.

[36] H. Zhao and R. Sakellariou. Advance reservation policies for workflows. In *12th Workshop on Job Scheduling Strategies for Parallel Processing*, 2006.

[37] Amazon EC2. `http://aws.amazon.com/ec2/`.

[38] Final report. teragrid co-scheduling/metascheduling requirements analysis team. `http://www.teragridforum.org/mediawiki/images/b/b4/MetaschedRatReport.pdf`.

[39] Parallel workloads archive. `http://www.cs.huji.ac.il/labs/parallel/workload/`.