# Candidacy Exam Proposal

Borja Sotomayor
Department of Computer Science
University of Chicago
borja@cs.uchicago.edu

November 24, 2008
Revision 33

**NOTE: I passed my candidacy exam on 11/25/08. At that time, my committee suggested some changes to my proposal (which are not included in this document). Nonetheless, this document paints a pretty accurate picture of where my dissertation research is headed.**

## Contents

# 1 Problem Statement

The need for computational resources has, over the years, become a fundamental requirement in both science and industry. In many cases, this need is transient: a user may only require computational resources for the duration of a well-defined task. For example, a scientist could require a large number of computers to run a simulation for just a few hours, but might not need those computers at any specific time (as long as they are made available in a reasonable amount of time). A college instructor may want to make a cluster of computers available to students during the course's lab sessions, at very specific times during the week, and with a specific software configuration. A telecommunications company could posses an existing infrastructure that hosts a number of websites, but may need to supplement that infrastructure with additional resources during periods of unforeseen increased web traffic, meaning those resources have to made available right away with very little advance notice.

These transient resource usage scenarios pose the problem of how to *provision shared computational resources efficiently*. This problem has been studied for decades, resulting in approaches that tend to be highly specialized to specific usage scenarios. For example, the problem of how to run multiple jobs on a shared cluster has been extensively studied, resulting in job management systems systems like Torque/Maui [21], Sun Grid Engine[1], LoadLeveler[2], and many others, that can queue and prioritize job requests efficiently (in these systems, efficiency is defined in terms of a variety of metrics, including waiting times and resource utilization). Such a system would meet the requirements of the scientist wanting to run simulations during a few hours but, on the other hand, the college instructor and the telecommunications company mentioned above would be ill-served by a job management system and the efficiency metrics typically used in job management. Conversely, other resource provisioning approaches are not particularly well suited for job-oriented computations (this point will be explored in greater detail in the next section).

Thus, there is no general solution that can provision resources meeting the requirements of different usage scenarios simultaneously, such as those mentioned above, reconciling the different measures of efficiency in each scenario. More specifically, we can constrain our discussion to the combination of *best-effort* resource requirements, where a user needs computational resources but is willing to wait for them (possibly setting a deadline), and *advance reservation* resource requirements, where the resources must be available at a specific time. In the former, efficiency is typically measured in terms of waiting times (or similar metrics such as turnaround times or slowdowns) or throughput, while the latter is usually concerned with providing the requested resources at exactly the agreed-upon times without interruption, and both are concerned with maximizing the use of hardware resources and possibly monetary profit. Although both best-effort and advance reservation provisioning have been studied separately, the combination of both is known to produce utilization problems (discussed in the next section) and is discouraged in practice.

In my research I seek to develop a resource provisioning model and architecture that can support multiple resource provisioning scenarios efficiently and simultaneously, with an initial focus on the best-effort and advance-reservation cases mentioned above, and arguing in favour of a *lease-based* model, where leases are implemented as *virtual machines* (VMs). This model must meet the following goals:

$G_1$ **Provide an abstraction focused solely on resource provisioning**
Although the lease abstraction has been used in multiple fields of computer science, most notably networking, there is no universally accepted definition of "lease". However, leases generally always provide an abstraction for, first and foremost, provisioning a resource (bandwidth in networks, raw hardware resources in datacenters, etc.) operated by a lessor (or *resource provider* and provided to a lessee (or *resource consumer*), with relatively few restrictions on how the provisioned resources can be used. So, when proposing a lease-based model, the implied goal is that resource consumers will be able to use a general-purpose resource provisioning abstraction (i.e., not one that is coupled to a particular use case).

$G_2$ **Provision hardware, software, and availability**

[1] `http://gridengine.sunsource.net/`
[2] `http://www.ibm.com/systems/clusters/software/loadleveler.html`

Resource provisioning can encompass three dimensions: hardware resources, the software available on those resources, and the time during those resources must be guaranteed to be available. A complete resource provisioning model must allow resource consumers to specify requirements across these three dimensions, and the resource provider to efficiently satisfy those requirements. As stated earlier, my work focuses on best-effort and advance reservations availability requirements.

$G_3$ **Reconcile requirements of different types of leases**
Best-effort and advance reservation provisioning have different measures of efficiency and, in some cases, these measures will be in conflict. For example, accepting advance reservations leases unconditionally may delay or even preempt best-effort leases but, on the other hand, a policy of not allowing best-effort leases to be delayed or preempted may reduce the number of advance reservations that can be accepted. Reconciling these measures of efficiency requires developing scheduling algorithms capable of combining both types of leases, and potentially others, and policies that can guide the scheduling decisions based on the goals and requirements of the resource provider. Taking into account the different overheads of virtual machines adds an additional layer of complexity to the problem of scheduling VM-based leases.

$G_4$ **Model virtual resources accurately and schedule them efficiently**
The choice of virtual machines to implement leases requires modelling virtualized resources and operations on those resources. In particular, using virtual machines involves different types of overhead (most notably the overhead of transferring virtual machine images, and the overhead of suspending and resuming virtual machines) that must be accurately modelled so they can be taken into account when scheduling virtual machines.

Summing up, the main contribution of my dissertation will be a resource provisioning model that uses leases as a fundamental abstraction and virtual machines as an implementation vehicle. This contribution can be further divided into a formal specification of lease terms, a model of virtualized resources, a lease management architecture (including scheduling algorithms and policies), and metrics of efficiency for heterogeneous workloads combining multiple types of leases. As a technological contribution, my dissertation will also provide an open-source reference implementation of the lease management architecture, capable of operating in simulation or on real hardware.

The remainder of this document is structured as follows: Section 2 presents existing work on resource provisioning, and motivates my dissertation work by explaining how existing approaches do not meet *all* four goals ($G_1, G_2, G_3, G_4$) mentioned above. Next, Section 3 briefly describes the current status of my work, followed by a description in Section 4 of the remaining work that I propose to do for my dissertation. Section 5 describes the research methodology that will be followed in the remaining work. Finally, Section 6 concludes by describing the applications of a VM and lease-based resource provisioning model.

# 2 Related Work

Many solutions to the resource provisioning problem have been researched and developed over time. The ones most related to my work fall into three broad categories: virtual machine-based approaches, lease-based approaches, and job-based approaches. This section describes relevant work in each of these categories and shows how no single solution solves all the issues I address in my dissertation, although some authors do propose solutions that can be leveraged in the more general model I propose.

## 2.1 VM-based approaches

My work argues in favour of using virtual machines as a vehicle for implementing leases. Several groups have explored the use of virtual machines as a resource provisioning mechanism, sometimes providing lease-like semantics. This section starts with an overview of the benefits and disadvantages of virtual machines followed by a discussion of existing VM-based resource provisioning work, divided into those that propose using virtual machines to create "virtual clusters" and those that provide lease-like semantics on large datacenters.

### 2.1.1 Virtual machines

Virtual machines are an appealing vehicle for resource management because they can be used to provision hardware, software, and availability ($G_2$):

**Hardware resources** Virtual machines can be mapped to all or part of a physical node's hardware resources, allowing users to request fine-grained resource allocations. Additionally, virtual machines have the added property of allowing these allocations to be strictly enforceable.

**Software environment** A virtual machine can encapsulate a custom software environment, allowing users to support existing applications without modifying them. Resource providers can also allow users to have administrative privileges within their VMs, with a reduced risk of malicious use thanks to the security and isolation properties of VMs. Additionally, popular VM systems, such as Xen and VMWare, primarily support the x86 architecture, facilitating virtualization of existing x86 compute resources. Although these systems require the VMs to use the x86 architecture too, a model based on VMs could easily support additional architectures once the VM vendors supported them.

**Availability** VMs can be suspended, potentially migrated, and resumed without modifying any of the applications running inside the VM. We have shown [44] that this capability allows us to efficiently combine leases with best-effort and advance reservation availability periods ($G_3$). Although there are other mechanisms to suspend/resume/migrate computation (such as checkpointing and preempting schedulers, described below), VMs provide a more versatile solution because they do not require applications, or even the OS running inside the VM, to be checkpointing-aware.

Although an attractive option, virtual machines also raise additional challenges ($G_4$) related to the overhead of using VMs:

**Preparation overhead** When using VMs to implement leases, a VM disk image must be either prepared on-the-fly or transferred to the physical node where it is needed. Since a VM disk image can have a size in the order of gigabytes, this preparation overhead can significantly delay the starting time of leases. This delay may, in some cases, be unacceptable for advance reservations that must start at a specific time.

**Runtime overhead** Once a VM is running, scheduling primitives such as checkpointing and resuming can incur in significant overhead since a VM's entire memory space must be saved to disk, and then read from disk. Migration involves transferring this saved memory along with the VM disk image. Similar to deployment overhead, this overhead can result in noticeable delays.

### 2.1.2 Virtual clusters

Several groups have explored the use of VMs to create "virtual clusters" on top of existing infrastructure. Nishimura et al.'s [32] system for rapid deployment of virtual clusters can deploy a 190-node cluster in 40 seconds. Their system accomplishes this 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; e.g., consider highly specialized scientific environments where installable binary packages may not be readily available) but does provide a faster alternative to VM image deployment if the installation time is short enough. Yamasaki et al. [49] 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. This 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.

The Nimbus toolkit[3] [22] has the ability to deploy "one-click" virtual clusters [9, 25] in sites with different software and network configurations, eliminating the need to manually adapt virtual machine images each time they are deployed in a new site (where, for example, the NFS server might have a different address, or services on the network might expect a digital host certificate signed by the local Certificate Authority). This is accomplished by using a standalone context broker that *contextualizes* disk images to work in a specific site.

Fallenbeck et al. [5] extended the Sun Grid Engine 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. [4] 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.

Walters et al. [48] have proposed the use of a new VM-centric job scheduler, called UBIS, capable of scheduling both traditional batch jobs and high-priority interactive jobs. This is accomplished by using the suspend/resume capability of virtual machines to preempt running batch jobs and accommodate incoming requests for interactive jobs. The UBIS scheduler not only facilitates support for interactive jobs, it also accomplishes impressive improvements (up to 500%) in resource utilization and response time for batch jobs. However, it does not support advance reservation of resources, instead focusing on supporting interactive jobs with near-immediate resource requirements, which simply allows the UBIS scheduler to perform preemption operations when an interactive job is requested. Supporting advance reservations in such a way that starting times can be guaranteed would require modelling this overhead and scheduling the preemption operations to finish before the start of a reservation.

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 [38, 37]), automatic configuration and creation of VMs (In-VIGO [1] and VMPlants [27]), and communication between a virtual cluster scheduler and a local scheduler running inside a virtual cluster (Maestro-VC's two-level scheduling [24]). However, they do not explore workloads that combine best-effort and advance reservation requests, nor do they schedule deployment overhead of VMs separately.

In general, all these solutions use virtual machines to great effect, addressing goal $G_2$ and, to a certain extent, $G_4$. However, all of them focus on provisioning resources for batch jobs (not providing a general provisioning abstraction, $G_1$) and focus on a single availability scenario (mostly the execution of batch jobs on a virtual cluster, which makes $G_3$ moot), except for Walters et al., who consider workloads combining both batch jobs and interactive jobs with near-immediate availability requirements. As far as $G_4$, only Nishimura et al. and Yamasaki et al. model and schedule the deployment overhead of virtual machines, while other groups either assume that this overhead does not exist (e.g., by assuming that VM disk images are predeployed) or can be ignored.

### 2.1.3 Datacenter-based solutions

Whereas the above solutions focus on creating virtual clusters, mostly for the purposes of job-based batch processing, other solutions focus on providing lease-like semantics where resource providers manage a datacenter and allow resource consumers to lease parts of it using virtual machines; a virtual cluster would be just one possible application of what the resource consumers could do with their virtual machines. In the server hosting arena, datacenters with virtualized resources have been a popular option for several years as a way of providing long-term resources leases where clients are given complete control over the leased resources, but without requiring a dedicated server per client. These are a popular option for hosting web/mail/DNS/etc. servers at a low price, but typically require leases with a minimum duration of a month. More recently, Amazon's EC2[4] introduced the notion of *cloud computing*, where virtual machines are dynamically provisioned immediately with customized software environments and charging for use by the hour. Eucalyptus

[3] Previously known as the Virtual Workspaces Service [23]

[4] `http://aws.amazon.com/ec2/`

[33] and Nimbus both provide an open-source alternative to Amazon's EC2, using the same web services interface and providing similar functionality.

Since this datacenter-based approach typically involves managing a large amount of virtual and physical servers, in the order of hundreds or thousands, efficiently managing the *virtual infrastructure* in the datacenter becomes a major concern. Several solutions, such as VMWare VirtualCenter, Platform Orchestrator, Enomalism, or OpenNebula have emerged to manage virtual infrastructures, providing a centralized control platform for the automatic deployment and monitoring of virtual machines (VMs) in datacenters. These solutions excel at providing users with exactly the software environment they require, and most provide a large number of hardware options. However, they depend on an immediate provisioning model, where virtualized resources are allocated at the time they are requested, without the possibility of requesting resources at a specific future time and, at most, being placed in a simple first-come-first-serve queue when no resources are available.

Since the workloads in datacenters typically involve servers running for long periods of time (in the order of months) with variable resource requirements, several groups have looked into the problem of how to use fewer physical servers by *consolidating* multiple virtual servers into single physical machines. This is a challenging problem that involves characterizing server workloads, predicting future resource demand [2], and then using this information to consolidate multiple servers on a single machine in such a way that the probability of breaching existing service-level agreements (SLAs) is minimized. This consolidation can be done when processing the requests for new servers (static consolidation) or it can be done while the servers are running (dynamic consolidation [2, 30]), typically by leveraging the live migration capability of virtual machines to optimize the mapping of VMs to physical machines.

Although all these solutions use a general lease-like abstraction ($G_1$) that allows users to request both hardware and a specific software environment ($G_2$), the lease terms are limited to just immediate availability; there is no possibility of requesting resources in advance or queuing requests, meaning these solutions have no need to address $G_3$. Nimbus, Eucalyptus, OpenNebula, and Enomalism all use a basic resource model that does not explicitly schedule VM overhead ($G_4$). Since the other cited work is closed-source and not peer-reviewed, it is hard to assess to what extent it addresses $G_4$.

## 2.2 Lease-based approaches

A purely lease-based approach to resource provisioning has been proposed by Grit et al. [14] and other members of Jeff Chase's research group [20, 36]. However, their work focuses mainly on leases in *federated* systems managed by their ORCA and Shirako systems. In such a system, resource providers can donate part of their resources to a broker (or multiple brokers) which can, in turn, give resource consumers "tickets" redeemable for actual resources when presented to a resource provider. Federation of leases across multiple sites is outside the scope of my work, which focuses on how resources are managed inside the resource provider. In fact, in the ORCA architecture, my work would be a resource provisioning "actor", the internal workings of which are supposed to be independent of ORCA, although the resources are assumed to be partitionable.

Most of their work uses virtual machines, managed by their Cluster-On-Demand system, as an example of a partitionable resource (although they emphasize that their work is applicable to any partitionable resource), and recent work has focused on how to enable batch job execution within their architecture [13, 15]. However, their model assumes that any overhead involved in deploying and managing the VM will be deducted from the lease's availability, instead of scheduling it separately ($G_4$). Additionally, despite providing a well thought-out leasing abstraction ($G_1$ and $G_2$), their work lives in a different level from mine (federated vs. local)

## 2.3 Job-based approaches

In science and academia and, to some extent, in industry, resource provisioning has been mostly tied to running *jobs* and many job schedulers have been developed over the years, such as Maui[5] [21], Moab[6], LSF[7], LoadLeveler[8], PBS Pro[9], and SGE[10]. However, job-based systems provision resources as a side-effect of having to run a job. So, although these systems can support both best-effort and advance reservation provisioning, resource consumers are required to interact with those resources using the job abstraction (i.e., goal $G_1$ is not met). Additionally, these systems include limited support for custom software environments ($G_2$), typically limiting resource consumers to whatever software environment happens to be available on the hardware resources being provisioned. The only exception is Moab, which provides limited support for starting up a virtual machine encapsulating the software environment required by the job. However, Moab only allows access to a limited number of VM-based environments, which still require considerable software contextualization on the part of the cluster administrator, and does not address the overhead of setting up those VMs ($G_4$)

Nonetheless, job scheduling has driven a considerable amount of research and led to important algorithms and results relevant to best-effort scheduling, including how to schedule advance reservations alongside best-effort workloads. Job schedulers typically depend on queues to prioritize access to resources, using backfilling [28, 31, 6] to optimize queue ordering. When using backfilling, the scheduler can make reservations in the future for requests that cannot be scheduled immediately, allowing subsequent requests in the queue to "skip the queue", as long as they finish before the future reservations. The scheduling algorithms used in my work depend on backfilling, but extend it by leveraging the suspend/resume capability of virtual machines.

This remainder of this section describes how advance reservations are supported in job-based systems, and how they can result in utilization problems. Although preempting schedulers partially palliate the utilization problems of advance reservations, they do not fully address some of the goals in my dissertation. Finally, I discuss multi-level scheduling solutions which use job-based systems purely as a resource provisioning tool (which would meet goal $G_1$), sidestepping the job abstraction and using other provisioning abstractions on the resources

### 2.3.1 Advance reservations in job-based systems

Although job schedulers can schedule advance reservations alongside best-effort workloads and, arguably, could be used to implement best-effort and advance reservation leases, these advance reservations fall short in several aspects. First of all, they are constrained by the job abstraction which, as described above, does not meet some of the goals I outlined for my dissertation. More specifically, when a user makes an advance reservation in a job-based system, the user does not have direct access to those resource but, rather, is allowed to submit jobs to them. For example, PBS Pro creates a new queue that will be bound to the reserved resources, guaranteeing that jobs submitted to that queue will be executed on them (assuming they have permission to do so). Maui and Moab, on the other hand simply allow users to specify that a submitted job should use the reserved resources (if the submitting user has permission to do so). There are no mechanisms to directly login to the reserved resources, other than through an interactive job, which does not provide unfettered access to the resources (i.e., no possibility of root access), or more ad-hoc methods, like requesting login privileges from the cluster administrator for the duration of the reservation.

Additionally, it is well-known that advance reservations lead to utilization problems [8, 40, 41, 29], caused by the need to vacate resources before a reservation can begin. Unlike future reservations made by backfilling algorithms, where the start of the reservation is determined on a best-effort basis, advance reservations introduce roadblocks in the resource schedule. Thus, traditional job schedulers are unable to efficiently schedule workloads combining both best-effort jobs and advance reservations ($G_3$).

[5] `http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php/`
[6] `http://www.clusterresources.com/pages/products/moab-cluster-suite/workload-manager.php`
[7] `http://www.platform.com/`
[8] `http://www.ibm.com/systems/clusters/software/loadleveler.html`
[9] `http://www.pbspro.com/`
[10] `http://gridengine.sunsource.net/`

However, advance reservations can be supported more efficiently by using a scheduler capable of preempting running jobs at the start of the reservation and resuming them at the end of the reservation. Preemption can also be used to run large parallel jobs (which tend to have long queue times) earlier, and is specially relevant in the context of urgent computing, where resources have to be provisioned on very short notice and the likelihood of having jobs already assigned to resources is higher. While preemption can be accomplished trivially by cancelling a running job, the least disruptive form of preemption is *checkpointing*, where the preempted job's entire state is saved to disk, allowing it to resume its work from the last checkpoint. Additionally, some schedulers also support job migration, allowing checkpointed jobs to restart on other available resources, instead of having to wait until the preempting job or reservation has completed.

However, although many modern schedulers support at least checkpointing-based preemption, this requires the job's executable itself to be checkpointable. An application can be made checkpointable by explicitly adding that functionality to an application (application-level and library-level checkpointing) or transparently by using OS-level checkpointing, where the operating system (such as Cray, IRIX, and patched versions of Linux using BLCR [16]) checkpoints a process, without rewriting the program or relinking it with checkpointing libraries. However, this requires a checkpointing-capable OS to be available.

Thus, a job scheduler capable of checkpointing-based preemption and migration could be used to address $G_3$, by checkpointing jobs before the start of an advance reservation, minimizing their impact on the schedule. However, the application- and library-level checkpointing approaches burden the user with having to modify their applications to make them checkpointable, imposing a restriction on the software environment being leases ($G_2$). OS-level checkpointing, on the other hand, is a more appealing option, but still imposes certain software restrictions on resource consumers. Systems like Cray and IRIX still require applications to be compiled for their respective architectures, which would only allow a small fraction of existing applications to be supported within leases, or would require existing applications to be ported to these architectures. This is an excessive restriction for supporting leasing, given the large number of clusters and applications that depend on the x86 architecture. Although the BLCR project does provide a checkpointing x86 Linux kernel, this kernel still has several limitations, such as not being able to properly checkpoint network traffic, and not being able to checkpoint MPI applications unless they are linked with BLCR-aware MPI libraries.

An alternative approach to supporting advance reservations was proposed by Nurmi et al. [34], which introduced "virtual advance reservations for queues" (VARQ). This approach overlays advance reservations over traditional job schedulers by predicting the time a job would spend waiting in a scheduler's queue, and submitting a job (representing the advance reservation) at a time such that, based on the wait time prediction, the probability that it will be running at the start of the reservation is maximized. Since no actual reservations can be done, VARQ jobs can run on traditional job schedulers, which will not distinguish between the regular best-effort jobs and the VARQ jobs. Although this is an interesting approach that can be realistically implemented in practice (since it does not require modifications to existing scheduler), it still depends on the job abstraction ($G_1$).

Hovestadt et al. [18, 17] propose a planning-based (as opposed to queuing-based) approach to job scheduling, where job requests are immediately planned by making a reservation (now or in the future), instead of waiting in a queue. Thus, advance reservations are implicitly supported by a planning-based system. Additionally, each time a new request is received, the entire schedule is reevaluated to optimize resource usage. For example, a request for an advance reservation can be accepted without using preemption, since the jobs that were originally assigned to those resources can be assigned to different resources (assuming the jobs were not already running). Although this approach is promising, and is arguably better in qualitative terms to a queuing-based approach, the authors show no quantitative results comparing their approach to a queue-based system or to a checkpointing-capable system.

### 2.3.2 Hierarchical/multi-level scheduling

In a hierarchical, or multi-level, scheduling model, a scheduler allocates resources that will be managed by a different scheduler. This approach has been widely used in the context of OS process scheduling, where one scheduler is responsible for allocating and managing heavy processes, while a separate scheduler, inside the process or as part of the OS, manages the threads more efficiently than the heavy process scheduler. Since

job-based systems tightly couple job execution to resource provisioning, multi-level scheduling solutions have emerged to circumvent this coupling: a job-based system is used to provision the resources but, instead of "running a job", the provisioned resources are then managed by a different scheduler, which can use different provisioning abstractions. Thus, multi-level scheduling approaches could be used to meet goal $G_1$, allowing resource providers to lease resources without having to shift from a job-based system to a purely lease-based resource manager.

The Condor scheduler's "glidein" mechanism [12] was the first to apply this model on compute clusters, using existing job schedulers to provision resources which would then be managed by an existing Condor pool. This is accomplished by starting (or "gliding in") Condor daemons on the provisioned resources. The MyCluster project [47] similarly allows Condor or SGE clusters to be overlaid on top of TeraGrid resources to provide users with "personal clusters". The Falkon task scheduler [19] can also be deployed through a GRAM interface on compute resources, and is specifically optimized to manage the execution of lightweight tasks, typically in a workflow managed by the Swift [51] system. Virtual workspaces [23, 9] also follow a multi-level scheduling approach by allowing users to create a workspace (represented as either a virtual machine or a dynamically-created UNIX account) on a remote site through a Virtual Workspace Service (VWS), and then allowing the user to access that workspace directly, and not through the VWS. The Workspace Pilot furthermore allows a job scheduler to allocate resources for the VWS, which are then managed by the VWS to create a virtual workspace on those resources[10].

By using a multi-level scheduling approach, Condor, MyCluster, Falkon, and the VWS can use their respective provisioning abstractions, sidestepping the site's job scheduling entirely. The first level of these multi-level scheduling solution approximates leasing, since it focuses only on resource provisioning. However, it is still limited to requesting the availability periods supported by job schedulers (best-effort jobs or advance reservations), and does not allow users to access them directly. Condor still requires the user to access the resources through Condor job submissions; similarly, MyCluster requires Condor or SGE jobs to be used, and Falkon requires computation to be expressed as tasks. Even if we overcame this issue (e.g., in theory, a job scheduler could be used to provision resources that could then be used to run an SSH server that allows the user to log into the provisioned resources), there is no support for deploying custom software environments.

## 2.4 Summary

Existing approaches to resource provisioning can be broadly categorized into three types: those that depend on virtual machines, those that use a lease abstraction, and those that use a job abstraction. Although many impressive solutions and results have been achieved in each of these areas, no single solution manages to address the four goals described at the beginning of this document. More specifically, there are two common failings:

**Only provisioning hardware, and neglecting software ($G_2$).** Solutions that do not use virtual machines typically focus on provisioning hardware and availability, limiting the choice of software to whatever software is installed on the hardware resources. Even though this software environment might be the result of a consensus between all the users or that hardware, and software package managers can be used to switch between a number of software environments, it still precludes users from getting fully custom software environments. Of the approaches described above, job-based systems are particularly limited in this respect, even when using multi-level scheduling to meet goal $G_1$ and checkpointing-aware preempting schedulers to meet goal $G_3$ since, as indicated above, using OS-level checkpointing further constraints the choices of software environment.

**Limited application of virtual machines.** Using virtual machines greatly facilitates provisioning fine-grained hardware allocations and custom software environments ($G_2$), and we have shown it to be an effective tool to combine best-effort and advance reservation availability requirements ($G_3$). Furthermore, some approaches use lease-like semantics ($G_1$) that do not tie resource consumers to using those resources for a specific purpose (such as job execution). However, most of these approaches use a basic resource model that neglects the overhead of deploying and running VMs and, even when deployment

overhead is modelled and taken into account for scheduling, as Nishimura et al. and Yamasaki et al. do, they focus only on immediate availability, and do not support workloads combining best-effort and advance reservation leases.

# 3 Status

At the time of writing this document, I have made the following progress on meeting the four goals identified at the beginning of this document:

$G_1$ **Provide an abstraction focused solely on resource provisioning**
My work is based on the lease abstraction, and I defined basic terms for best-effort and advance reservation leases [11, 44].

$G_2$ **Provision hardware, software, and availability**
The lease terms in my model encompass hardware, software, and availability, with the current focus on best-effort and advance reservation availability. I designed [44] and implemented[11] a lease management architecture called *Haizea*, capable of scheduling leases, either in simulation of by managing virtual machine-based leases on real hardware [46]. Virtual machines were chosen as an implementation vehicle since, as described in Section 2, they are capable of provisioning hardware, software, and availability.

$G_3$ **Reconcile requirements of different types of leases**
Produced simulation-based results showing that the Haizea lease manager is capable of scheduling best-effort and advance reservation leases together more efficiently than previous approaches [45, 44].

$G_4$ **Model virtual resources accurately and schedule them efficiently**
Defined a basic model for virtualized resources, with emphasis on the overhead of managing those resources [43, 42].

The remainder of this section provides a summarized version of the main results outlined above.

We defined 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" [44]. The terms must 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. Since previous work and other authors already explore lease terms for hardware resources and software environments [11, 25], my recent focus has been on the availability dimension of a lease.

Thus, at this point, I consider the following simple availability terms for a lease:

- *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.

Furthermore, I have so far made the simplifying assumption that all advance reservation leases are nonpreemptable and all best-effort leases are preemptable and 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

[11] `http://haizea.cs.uchicago.edu/`

as priorities, would result in a decision not to preempt). The next section will explain how this assumption will be removed.

Our resource model currently considers that we manage $W$ 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 assume that all disk images required to run virtual machines 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. 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 before the VM can start. When transferring a disk image to multiple nodes, we use multicasting and model the transfer time as $\frac{s_I}{B}$. 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.

The Haizea lease management architecture is designed to process lease requests and schedule virtual machines to satisfy the lease's requirements. To do this, Haizea's scheduler uses both classical backfilling algorithms [31] and the suspend/resume/migrate capability of VMs, allowing best-effort leases to be preempted, and potentially migrated, to make room for advance reservation requests. Additionally, Haizea explicitly schedules the preparation and runtime overhead of VMs to guarantee that the lease terms are met. More specifically, for advance reservation leases, disk image transfers are scheduled to arrive before the start of the lease and, if best-effort leases are being suspended before the AR lease, the suspensions are also scheduled to complete before the start of the AR lease. For all types of leases, the schedulers attempts to minimize the number of disk image transfers by reusing disk images in the physical nodes when possible.

Haizea has been implemented, carefully documented, and released under an open-source Apache 2.0 License[12], and is available for download at `http://haizea.cs.uchicago.edu/`. The latest version, Technology Preview 1.2, can run scheduling simulations (like the ones shown in some of my papers [42, 45, 44]) and, in combination with the OpenNebula virtual infrastructure manager[13], can create and manage VM-based leases on clusters using the Xen or KVM hypervisors.

Our main results so far [44] have shown that, when using workloads that combine best-effort and advance reservation lease requests, a VM-based approach with suspend/resume/migrate can overcome the utilization problems typically associated with the use of advance reservations. 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. Measuring the wait time and slowdown of best-effort leases shows that, although the average values of these metrics increase when using VMs, this effect is due to short leases not being preferentially selected by Haizea's backfilling algorithm, instead allowing best-effort leases to run as long as possible before a preempting AR lease (and being suspended right before the start of the AR). In effect, a VM-based approach does not favor leases of a particular length over others, unlike systems that rely more heavily on backfilling. Our results have also shown that, although supporting the deployment of multiple software environments, in the form of multiple VM images, requires the transfer of potentially large disk image files, this deployment overhead can be minimized through the use of image transfer scheduling and caching strategies.

[12] `http://www.apache.org/licenses/LICENSE-2.0.html`
[13] `http://www.opennebula.org/`

# 4 Research Plan

My work so far has addressed the resource provisioning problem by tackling the four goals addressed at the beginning of this proposal, without compromising any goal in favour of another one. However, there are still some gaps and assumptions in my work that need to be resolved:

(A) *Using a constrained resource model.* So far, all my results have been simulation-based and, although they have been useful in observing the long-term impact on certain performance metrics, they require using a very constrained resource model, since I did not have an testbed of multiple nodes to experiment on (thus $G_4$ was not being fully addressed). Most notably, I currently assume that only one VM can run on each physical node and that memory state files (resulting from suspending a virtual machine) are saved to that node's local filesystem. Another way of stating this assumption is that my model currently assumes no possibility of contention for resources during a suspend or a resume operation (since a single VM has exclusive use of a physical node). This is a very limiting assumption in practice, given the widespread availability of global filesystems and given the increasing popularity of multicore machines which facilitate running more than one VM on a single node.

(B) *There is no lease admission control.* No lease requests are rejected, except AR leases where no resources are available for the lease. This "accept all" policy has been acceptable for experiments so far, where the AR workloads were designed not to saturate the simulated cluster, and were generated according to different parameters, allowing us to measure the impact of each parameter. However, in practice, this policy does not prevent resource consumers from always requesting advance reservations, sidestepping the queue for best-effort requests. This can lead to an inefficient use of resources, since our results have shown that, as the number of ARs accepted increased, so did the waiting time and slowdown of best-effort leases, since the scheduler is faced with a more inflexible schedule. Thus, $G_3$ is not met unless there is some form of admission control in my model.

(C) *Scheduling algorithm is not adaptive.* Although the scheduling algorithms I have implemented can *react* to variations in resource usage (more specifically, the scheduler can reevaluate the schedule upon an early termination of a lease), the scheduler does not dynamically *adapt* to these variations or to changes in user behaviour. This is not an issue when simulating existing workloads, since the workload never changes throughout the experiment[14]. However, in practice, users will behave differently depending on how the scheduler handles their requests, and a scheduler must be able to adapt to those changes.

(D) *I assume that best-effort leases are preemptable and AR leases are not.* This was made as a simplifying assumption, but it does not fully model reality, where leases will not be just "preemptable" or "not preemptable"; rather, preemptability should be derived from a variety of factors (the same way, for example, that the priority of a batch job is computed based on multiple factors).

To complete my dissertation research, these issues must be resolved. I propose doing so by meeting the following milestones:

- Refining the resource model to more accurately represent leased virtual resources.
- Researching policies for lease admission and lease preemption.
- Researching adaptive scheduling strategies for advance reservations.

The remainder of this section describes each of these tasks in more detail, specifying what work each milestone would involve, and the results that should be expected at the end of each milestone.

[14] For avoidance of doubt: In my simulation experiments I do not assume foresight of the entire workload (i.e., at any point in time, the scheduler has no knowledge of what requests will be arriving in the future).

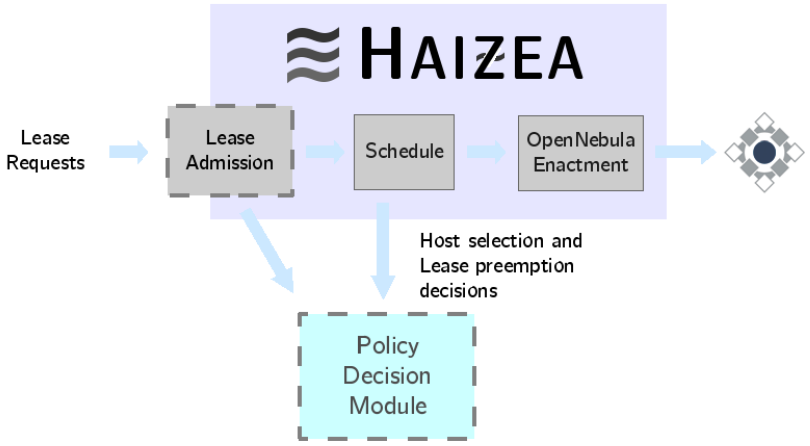


**Figure 1:** The Haizea architecture is still missing a lease admission component and a policy-driven decision module

## 4.1 Refining the resource model

To address (A), I will have to conduct experiments on real hardware to obtain a more accurate model of how virtual resources behave under a variety of conditions, specially those where contention for resources is likely to happen. I am already working on this by using Haizea and OpenNebula to deploy best-effort leases and an advance reservation leases on a cluster, and measuring the times required to perform operations such as suspending/resuming/migrating VMs with local and global filesystems, using leases of varying sizes, and measuring the impact these operations have on other leases running on the cluster. This refinement of the resource model is not a strong research contribution but, rather, a prerequisite before moving on to the other milestones, as it allows us to show that are simulations are grounded on a model that has been verified on physical hardware.

A draft of our results will be submitted for publication in December 2008.

## 4.2 Policies for resource leasing

To address (B) and (D), the Haizea architecture needs to be extended so that decisions on lease admission and resource preemption are configurable, instead of being hardcoded. As a technical prerequisite, this will require factoring out the admission control and resource preemption decisions out to a *policy-driven decision module* (see Figure 1), where resource providers can specify policies to guide these decisions. For accepting/rejecting leases, these policies will guide whether a lease request should be passed on to the lease scheduler, or rejected outright (e.g., a policy could set quotas on how many ARs a user can request). For resource preemption, given a set of leases that may have to be preempted to accommodate another lease, policies would allow that set of leases to be ordered from "most preemptable" to "least preemptable" (e.g., a policy could determine that the shorter the duration of the lease, the more desirable it is to preempt, since it will be easier to reschedule that a long lease). The policy module will also allow multiple policies, each with a different weight (specified by the resource provider), to be used when making an admission control or resource preemption decision.

Once this component has been added to Haizea, I believe the interesting research questions arise in addressing (B), whereas addressing (D) just provides resource providers with a greater degree of control over what leases can and cannot be preempted. As stated earlier, best-effort and advance reservation leases have different measures of efficiency that can conflict with each other. Lease admission can have a great impact on these measures: accepting more advance reservations will penalize best-effort leases and increases

the chance of two ARs being in conflict, while accepting more *non-preemptible* best-effort leases (where resources are still provisioned on a best-effort basis but, once they are allocated, we want to provide the resource consumer with a guarantee that those resources will be available) will reduce the number of advance reservations that can be accepted. Although policies for best-effort workloads (particularly jobs, and how to assign priorities to them) have been studied thoroughly, much less work has been done on policies that must deal with workloads combining both best-effort and AR requests. In particular, I am interested in the following questions:

1. How can users have an incentive to request one particular type of lease? In particular, in the presence of AR leases, how can users still be incentivated to requests best-effort leases? More generally, what is the behaviour of users when they are given the option of requesting advance reservations?
2. Given that these policies can suspend/resume/migrate virtual machines (operations with both a benefit and a cost), how would these same policies perform on systems that do not operate on VMs?

To answer these questions, I propose doing the following:

- Add a policy decision module to Haizea.
- Develop a set of user behaviour models. "Users" would have workloads they want to run, and will behave differently based on the responses they get back from the lease admission component. This user model will build upon existing user models (particularly those that are used when generating artificial job workloads) and on the usage patterns observed in existing systems where advance reservations are supported.
- Develop a set of policies for lease admission with the goal of maximizing resource utilization, while reconciling the different goals of best-effort and advance reservation leases. At this point, the policies will be static, and will not adapt to changes in user behaviour.
- Run experiments (see next section, *Research Methodology*), where different policies are matched up with different user behaviour models. The results should show how effective the policies are at satisfying the requirements of both best-effort and advance reservation leases in the presence of a particular type of user (or a mix of several types of users).

### 4.3 Adaptive scheduling for advance reservations

Variability in resource requests and actual resource usage by resource consumers is a fact of life, and a considerable amount of research has been done to address this variability in different systems. In job-based systems, users tend to request resources for a much longer duration than they actually need them, and much work has been done in accurately predicting job runtime . In VM-based data centers, users rarely use all the resources they are allocated, which has lead to the development of several algorithms for server consolidation [2, 30], which analyse and predict actual resource usage to "pack" several VMs into a single processor, increasing resource utilization while meeting service-level agreements with users. In lease-based systems, the problem of how to extend leases to accomodate changes in resource demands has also been addressed [20]. Although these existing solutions could be leveraged in Haizea to address (C), this would not be a research contribution.

On the other hand, adapting to variability in systems supporting both best-effort and advance reservation workloads has (to the best of my knowledge) not been studied. In particular, the problem becomes how to adapt to changes in user behaviour when they are given the ability to request advance reservations. So, while the previous milestone focused on "static lease admission" (the policies did not adapt to changes in user behaviour), this milestone would focus on "dynamic lease admission". The main question would be: how can we dynamically adjust a lease admission policy in a system supporting both best-effort and advance reservations leases to meet a target function, such as maximizing utilization, minimizing waiting times, or maximizing profit?

Answering this question will require extending the work from the previous milestone to add adaptability to the lease admission policies. Results would compare the previous policies to the new adaptable policies, showing how effective each is at meeting target functions such as those mentioned above.

# 5 Research Methodology

The general methodology to validate each of the proposed milestones will be to run experiments where a workload of leases must be scheduled and the values of several metrics are measured throughout the experiment and compared to a baseline. These experiments will be performed on real hardware and in simulation using Haizea. This section describes the workloads, metrics, and baselines I will use.

Although there are currently several repositories of batch job workloads, most notably the Parallel Workloads Archive [15], there are few publicly available advance reservation workloads or lease-based workloads. So far, my approach has been to take a workload of job submission requests from the Parallel Workloads Archive, treating that workload as a set of best-effort lease requests, and inserting an additional set of advance reservation requests. These advance reservation requests are artificially generated according to a number of well-defined parameters [44]. When running in simulation, a large number of requests (in the order of weeks or months) can be processed to observe the long-term effects of different modelling and scheduling decisions. When running on real hardware, shorter workloads (in the order of hours or days) are run to verify that (1) the simulations accurately represent reality and (2) to refine the resource model.

The metrics I intend to use measure how efficiently each type of lease is scheduled. For best-effort leases, let $t_a$ be the arrival time, or time when the lease request is submitted, $t_s$ be the start time of the lease, and $t_e$ be the time the lease ends. Based on these variables, I have so far used the following metrics:

**all-best-effort:** The time when the last best-effort lease is completed, or $\max(t_e)$. This metric provides a good measure of global utilization, by showing how fast a specific configuration of the scheduler can work through *all* the best-effort leases, although it is inadequate to analyse the effect on individual leases. Note that, in practice, this metric is normalized relative to some baseline value (e.g., the time required to complete all best-effort leases assuming no advance reservation leases are added to the workload).

**Wait time of best-effort requests:** The time a best-effort request must wait before it starts running, or $t_s - t_a$

**Bounded slowdown of best-effort requests [7]:** 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 [7].

In future experiments I intend to use metrics applicable to advance reservation leases, such as the number of leases accepted and rejected (which, so far, did not make sense since all ARs are accepted by Haizea). As mentioned in Section 1, each type of lease can have often-conflicting measures of efficiency, which will also necessitate designing a unified metric that shows not only how a single type of lease is performing but also how efficiently the system is handling multiple types of lease simultaneously. Additionally, I will also use metrics that measure the utilization of physical resources and, more specifically, what percent of the resources throughout an experiment are idle, running a VM, suspending, resuming, transferring a disk image, etc. This metric will be helpful in determining the cost of using the model I propose, by showing how many resources are spent on overhead activities that would not be necessary if not using VMs.

When analysing the values of these metrics in individual experiments, the baseline will be existing resource provisioning approaches, such as those described in Section 2, and, more specifically, those that can provide similar functionality to the one I propose. For example, since job-based systems can provide both best-effort and advance reservation provisioning, they provide a useful baseline when running workloads that combine best-effort and advance reservation leases. On the other hand, a datacenter-based solution would be an

[15] `http://www.cs.huji.ac.il/labs/parallel/workload/`

inadequate baseline as they do not support these workloads, but might be a good baseline when comparing policies for immediate reservation of resources.

# 6 Applications

Several applications stand to benefit from a lease-based model. Applications expressible as a workflow of small tasks have been shown to be more efficiently scheduled by a workflow engine operating over leased resources instead of through a job scheduler [39, 50]. Applications that currently rely on advance reservations [8, 52], such as grid applications requiring co-scheduling of resources across multiple sites, and urgent computing [35] applications could benefit from leasing mechanisms and policies that allow users to dynamically negotiate exact and urgent availability periods. Some applications, such as the STAR experiment[16], require complex software environments that are rarely available on grid sites (which must satisfy the software needs of multiple communities), and could benefit from a lease model that allows fully customized software environments to be dynamically instantiated on leased resources (the STAR application has been shown to benefit from this approach [26]).

More recently, the EU RESERVOIR project[17] is developing a model and architecture for open federated cloud computing[3], driven by the requirements of several business use cases, such as SAP systems, Software-as-a-Service providers, and virtualized data centers. Two of the primary requirements identified by the RESERVOIR project are "Automated and fast deployment", which my model attempts to optimize by explicitly scheduling deployment of VMs and reducing the overhead of these operations, and "dynamic elasticity", which adaptive scheduling (see Section 4.3) could contribute to.

# 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] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on Integrated Management*, pages 119–128, 2007.

[3] B.Rochwerger, D.Breitgand, E.Levy, A.Galis, K.Nagin, I.Llorente, R.Montero, Y.Wolfsthal, E.Elmroth, J.Caceres, M.Ben-Yehuda, W.Emmereic, and F.Galán. The RESERVOIR model and architecture for open federated cloud computing. *IBM Systems Journal*, Accepted for publication.

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

[5] 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.

[6] 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.

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


[16] `http://www.star.bnl.gov`
[17] `http://www.reservoir-fp7.eu/`

[8] 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.

[9] 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.

[10] T. Freeman and K. Keahey. Flying low: Simple leases with workspace pilot. In E. Luque, T. Margalef, and D. Benitez, editors, *Euro-Par*, volume 5168 of *Lecture Notes in Computer Science*, pages 499–509. Springer, 2008.

[11] 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.

[12] 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.

[13] L. Grit, D. Irwin, V. Marupadi, P. Shivam, A. Yumerefendi, J. Chase, and J. Albrecht. Harnessing virtual machine resource control for job management. In *Proceedings of the First Workshop on System-level Virtualization for High Performance Computing (HPCVirt)*, 2007.

[14] L. E. Grit. *Extensible resource management for networked virtual computing.* PhD thesis, Durham, NC, USA, 2007. Adviser-Jeffrey S. Chase.

[15] L. E. Grit and J. S. Chase. Weighted fair sharing for dynamic virtual clusters. In *SIGMETRICS '08: Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 461–462, New York, NY, USA, 2008. ACM.

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

[17] F. Heine, M. Hovestadt, O. Kao, and A. Streit. On the impact of reservations from the grid on planning-based resource management. In *Proceedings of the 5th International Conference on Computational Science (ICCS 2005), volume 3516 of Lecture Notes in Computer Science (LNCS*, pages 155–162. Springer, 2005.

[18] M. Hovestadt, O. Kao, A. Keller, and A. Streit. Scheduling in hpc resource management systems: Queuing vs. planning. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *JSSPP*, volume 2862 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2003.

[19] 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.

[20] 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.

[21] 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.

[22] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa. Science clouds: Early experiences in cloud computing for scientific applications. In *Cloud Computing and Applications 2008 (CCA08)*, 2008.

[23] 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.

[24] 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.

[25] K.Keahey and T.Freeman. Contextualization: Providing one-click virtual clusters. In *eScience 2008*, 2008.

[26] K.Keahey, T.Freeman, J.Lauret, and D.Olson. Virtual workspaces for scientific applications. In *SciDAC 2007 Conference*, 2007.

[27] 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.

[28] 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.

[29] 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.

[30] S. Mehta and A. Neogi. Recon: A tool to recommend dynamic server consolidation in multi-cluster data centers. *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, April 2008.

[31] 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.

[32] 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*, Washington, DC, USA, 2007. IEEE Computer Society.

[33] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Cloud Computing and Applications 2008 (CCA08)*, 2008.

[34] D. C. Nurmi, R. Wolski, and J. Brevik. Varq: virtual advance reservations for queues. In *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*, pages 75–86, New York, NY, USA, 2008. ACM.

[35] 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.

[36] L. Ramakrishnan, D. Irwin, L. Grit, A. Yumerefendi, A. Iamnitchi, and J. Chase. Toward a doctrine of containment: grid hosting with adaptive resource control. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 101, New York, NY, USA, 2006. ACM.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

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

[43] 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.

[44] B. Sotomayor, K. Keahey, and I. Foster. Combining batch execution and leasing using virtual machines. In *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*, pages 87–96, New York, NY, USA, 2008. ACM.

[45] B. Sotomayor, K. Keahey, I. Foster, and T. Freeman. Enabling cost-effective resource leases with virtual machines. In *Hot Topics session in ACM/IEEE International Symposium on High Performance Distributed Computing 2007 (HPDC 2007)*, 2007.

[46] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. Capacity leasing in cloud systems using the opennebula engine. In *Cloud Computing and Applications 2008 (CCA08)*, 2008.

[47] 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.

[48] J. P. Walters, B. Bantwal, and V. Chaudhary. Enabling interactive jobs in virtualized data centers. In *Cloud Computing and Applications 2008 (CCA08)*, 2008.

[49] 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.

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

[51] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde. Swift: Fast, reliable, loosely coupled parallel computation. *IEEE International Workshop on Scientific Workflows*, 2007.

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