Flying Low: Simple Leases with Workspace Pilot

Timothy Freeman[1], Katarzyna Keahey[1,2]

[1]Computation Institute, University of Chicago
[2]Argonne National Laboratory
{tfreeman,keahey}@mcs.anl.gov

Abstract. As the use of virtual machines (VMs) for scientific applications becomes more common, we encounter the need to integrate VM provisioning models into the existing resource management infrastructure as seamlessly as possible. To address such requirements, we describe an approach to VM management that uses multi-level scheduling to integrate VM provisioning into batch schedulers such as PBS. We then evaluate our approach on the TeraPort cluster at the University of Chicago.

1 Introduction

Resource leases – allowing a user to request direct access to resources rather than ask for a job to be run on those resources– are emerging as a fundamental abstraction of computing infrastructure. A lease may take the form of a static, long-term agreement with a hosting company, on-demand provisioning of a physical cluster partition with specified configuration as implemented by cluster-on-demand [1], or dynamically deploying a virtual machine for an hour on resources provided by Amazon's EC2 service [2]. A user can adapt such resource to his or her needs in a variety of ways: use it to support an interactive session, run computations requiring an application-specific scheduler, or support portability tests across a variety of environments. The need for resource leases rather than running jobs is exemplified by the widespread popularity of various "pilot job" approaches [3-6] that use batch scheduler installations on sites to deliver a lease rather than submit a job to that scheduler.

Virtual Machines (VMs) represent an ideal vehicle for implementing resource leases because of their isolation and enforcement properties. Among others, a VM configured by an application scientist can be deployed on many different sites without requiring the resource providers to understand the application and its dependencies, or integrate them into the configuration of their site [7, 8]. This makes VM-based resource provisioning attractive to both providers and consumers [9]. However, despite these advantages, VMs have not seen widespread adoption so far due to a relatively high barrier involved in adapting a site infrastructure for VM deployment. A solution is required that would allow sites to experiment with VM adoption without committing themselves to VM-based operations.

In this paper, we use the "pilot jobs" approach, combined with the VM management capabilities of the workspace service [10], to adapt local resource managers (LRMs), such as Torque [11] or SGE [12], for VM deployment. Our "workspace pilot" allows a resource provider to continue making its resources available to run jobs

via a batch scheduler within the model of operations prevalent today but also allows for the deployment of VMs as need arises. Further, since the workspace pilot requires no modifications to the LRM it extends, it enables non-invasive, easy adoption. We discuss the advantages and limitations of the proposed approach as well as the implementation details of the relevant parts of the workspace service. The workspace pilot has been deployed at the TeraPort cluster at the University of Chicago [13] using Torque as the LRM: we evaluate and discuss the its performance on that system.

This paper is structured as follows. Section 2 describes our approach and discusses its advantages and limitations. Section 3 describes the relevant implementation details. Section 4 contains the evaluation on the TeraPort cluster. Section 5 discusses the related work and we conclude in Section 6.

2 Approach

We use a multi-level scheduling approach, similar to that employed by Condor [3] glide-ins, to enable resource leasing with VMs. This approach relies on submitting a job to the LRM with a resource allocation request expressed in terms of duration and resources (such as number of nodes or memory). When scheduled, the job request results in the deployment of a "pilot program" that adapts the node for use within its framework and reports the availability of the node to an external framework (e.g. by joining a Condor pool). Our work leverages this mechanism to adapt physical resources for VM deployment, and then reports their availability for VM hosting to the Workspace Service which provisions VMs on it.

2.1 Overview of the Workspace Services

The virtual workspace services (VWS) [10] are a group of WSRF [14] services that allow an authorized remote client to deploy and manage workspaces (implemented as VMs). The *workspace factory* makes available the descriptions of lease types available on a specific site (e.g., what resources can be assigned to a VM). Using the factory, a client can deploy a workspace, which can be represented by a VM, a group of *n* homogeneous VMs (each associated with the same image and resource allocation -- memory, CPU, etc. available to the VM), or a group of heterogeneous VMs on a specified set of resources [15]. Once the workspace is deployed, the *workspace service* allows the client to access information related to each deployed VM. The client can also subscribe for notifications of events related to the VM lifecycle (e.g., it can be notified when a VM is deployed). The *workspace group* and *workspace ensemble services* allow a client to obtain information about and manage workspaces made up of homogeneous and heterogeneous VMs respectively.

Workspace services are intended to be deployed on a service node of a cluster (i.e., "gatekeeper", or headnode) and rely on a VM manager (workspace back-end) deployed on a set of worker nodes to carry out VM management requests. The workspace tools provide a default implementation of such manager (called "workspace default") that provides a simple, "greedy" mapping of workspaces to nodes. The VM

manager back-end could also be implemented by a datacenter technology or by the combination of an LRM and the workspace pilot described here.

2.2 Two-Level Provisioning

The approach described here assumes that VMs will be leased on a cluster equipped with *n* nodes each of which is configured such that they can serve both as job platform and a VM platform (e.g., Xen [16] nodes that have been booted into domain 0). Each node has access to local disk storage, which we use to store VM images. The nodes are managed by an LRM.

The objective of the "pilot program" submission is to obtain a time-constrained lease of a number of physical cluster nodes and adapt those nodes to make the deployment of VMs possible. We call such lease a *resource slot*. As shown below, a resource slot can support the deployment of potentially multiple VMs (*virtual resources*). We thus operate in a two-level provisioning model: resource slots are provisioned from the resource provider (i.e., the physical clusters) and then virtual resources are provisioned from those slots.

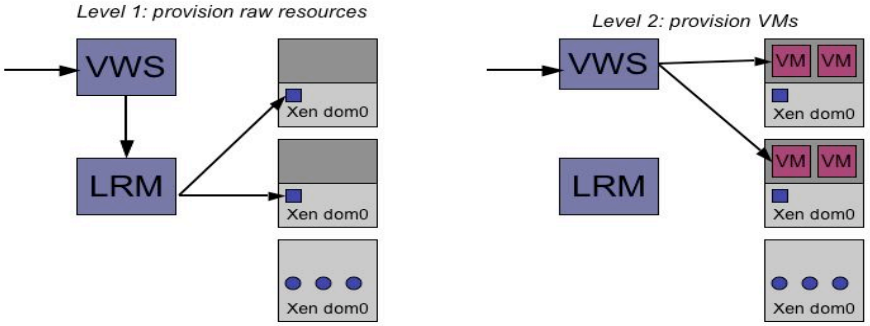

Figure 1: Two level scheduling: (1) the pilot adjusts the memory to obtain a resource slot, and (2) the obtained slots are used to schedule VMs.

The provisioning and deployment of resource slots takes place as follows. First, the workspace service submits a pilot job with requirements defining a resource slot to the LRM. The resource slot is defined in terms of duration, the number of nodes and the number of processors per node. The LRM queues and eventually executes the request. On execution, the pilot job adapts the platform for VM deployment (e.g., adjusts the memory of Xen domain 0 to allow for deployment of user domains). Then, the pilot notifies the workspace service that the resource slot has been obtained. When the resource slot terminates the pilot gracefully terminates the slot collaborating with the workspace service on cleanup actions.

The provisioning of virtual machines (the workspace default) operates as follows. On receiving a client request, the workspace service estimates the amount of physical resource needed to provide the resource allocation requested by the client. If the physical resource is already available (as a result of a prior LRM request) the required resource slot is returned, otherwise a resource slot is requested via the LRM. Once the slot is obtained, the workspace service maps the VMs onto the resource slot using

greedy algorithm and the client is notified of the availability of the VM. After the elapsed VM lifetime or upon client request the VMs are terminated.

Note, that the approach described here replaces the assumption that workspace deployment relies on a static set of physical resources with a dynamically provisioned (and dynamically released) set of physical resources. The Workspace Service may implement a variety of policies when requesting/releasing resources via the pilot and mapping VMs onto those resources, e.g. physical resources may be provisioned proactively or in direct response to a request, they may be overprovisioned (e.g., allow for renegotiation of slot duration), or already provisioned resource may be opportunistically used to schedule different workspaces. All these policies balance utilization costs versus flexibility, response time and request priority. The default policy (evaluated in this paper) favors resource utilization: we request exactly as many physical resources as needed to support an incoming workspace request. Further, the default policy always translates the requested resources into the smallest resource slot into which they fit so that it can be provisioned by the LRM, mapping one VM per node.

2.3 Leasing Resources with Workspace Pilot: Client's Viewpoint

Since the workspace pilot relies on the LRM to schedule physical resource leases, the lease semantics it can provide are limited by the LRM. In particular, if our default policy is followed (one physical lease gets requested per virtual lease) the lease semantics will follow the LRM policies exactly. As many sites today do not give users the ability to do advance reservations (even when this functionality is provided by the scheduler) due to policy issues, the leases provided by the workspace pilot are likely to be "best effort" i.e., they will provision the lease as resources become available.

Most scientific computations today are performed on integrated cluster infrastructure where nodes configured to perform specialized services (data access, compute nodes, worker nodes) collaborate on satisfying demands of specific computation. For a functional cluster to be deployed it is necessary to deploy all those nodes at once. Especially when the lease semantics are best-effort (i.e., we cannot time-synchronize that deployment), we need to provide mechanisms that will ensure that the VMs representing cluster nodes are deployed together. This role is fulfilled by the workspace group and ensemble services (see Section 2.1) that allow users to prepare groups of services to be deployed together.

We now describe an example of how various features of the workspace tools are used by a client in a simple deployment case. In order to deploy a complex cluster, e.g. an Open Science Grid (OSG) cluster consisting of a compute element (CE), a storage element (SE) and *n* worker nodes [17], a client performs the following actions. First, the client requests the creation a workspace (an "ensemble") consisting of n worker node VMs, and a VM representing the SE and CE each. As part of the creation request, the worker nodes request an IP on a private network while each of the service nodes requests a public IP assignment in addition to the private network IP assignment. Since the ensemble workspace is created in one request, the workspace service will allocate resources to host all the members of the ensemble and they all will be deployed at the same time. The final remaining step is to "contextualize" the

cluster (i.e., integrate deployment time configuration information into the cluster); this process exceeds the bounds of this paper and is described elsewhere [18].

3 Implementation

To implement the approach described above we developed the workspace pilot program, the workspace control program that integrates the VMs into the network fabric, and a simple LRM adapter.

3.1 The LRM Adapter

The implementation of the LRM adapter is composed of two components. The first is implemented by the workspace service and consists of control and monitoring interfaces to the LRM (e.g., *qsub* and *qdel* commands for Torque). We assume that the LRM may send a courtesy catchable signal before job termination (SIGTERM in Torque) which allows the pilot to implement graceful exit procedures upon receiving the signal. The second component (described in 3.2) is therefore implemented within the pilot program and consists of signal handlers.

3.2 The Workspace Pilot

The actions required to adapt a job platform for VM hosting depends on how much these two platforms differ. Our implementation assumes that the cluster which constitutes a job platform is installed with Xen and booted into domain 0 with maximum amount of memory given to each node. Therefore, to bridge the gap between the job platform and VM hosting platform we only have to reduce domain 0 memory so as to be able to host user VMs.

On deployment, the workspace pilot is given information about the expected duration of the slot and the requested resources. Based on this information the pilot reduces the domain 0 memory using the Xen balloon driver (this is a privileged operation which requires the pilot runs in to have sudo authorization). It is a matter of policy how the memory requirements of a VM are translated into the actual memory reduction: reserving more than absolutely necessary allows us to potentially schedule other VMs in the same slot but on the other hand leaving more memory in domain 0 can favorably impact guest performance [9] (the minimum is currently set to 100 MB). After the memory is adjusted, the pilot notifies the workspace service that the slot is ready. Under a normal set of circumstances the pilot is terminated either by a direct request from the workspace service or because the duration of the slot has expired. It then calls the “release slot” operation that will completely undo the effects of the “reserve slot”.

Occasionally the pilot program may receive a catchable signal (SIGTERM) from the LRM, e.g. if it has exceeded its allotted time, is being preempted or removed by the LRM, or due to a reboot action. The workspace pilot catches the warning signal

(which in effect offers a "grace period" before hard termination) and implements the following signal handler. It first notifies workspace service that it has received a preemption signal. It then waits for a portion of the "grace period" for the workspace service to clean up any running VMs. If the cleanup does not occur, the pilot destroys the VMs itself, restores the memory, and notifies the workspace service of the performed actions. To address the situation where a system administrator has to manually restore the nodes to a job platform we implemented a standalone "kill all" program (a direct call to the hypervisor to immediately destroy all local VMs and adjust the memory to release all available memory back to domain 0).

The workspace pilot program communicates with the workspace service via a configurable protocol by default relying on HTTP-based notifications (with SSH-based communications also possible). These notifications are time stamped so that they can be tracked by the workspace service for state recovery (e.g., in the event that the service itself recovers from failure).

3.3 The Nuts and Bolts of VM Deployment: Workspace Control Implementation

The workspace control program manages VMs on individual nodes based on commands received from the workspace service. Its primary functions are: (1) to start, stop and pause VMs, (2) to provide VM image reconstruction and management, (3) connect the VMs to the network, and (4) to deliver contextualization information [18].

To carry out VM image management functions, workspace control transfers VM images from a location within the site to the node on which it executes. The workspace service allows clients to request VM image reconstruction from disk partitions cached on the site, e.g. if a large disk partition is frequently used by images deployed on that site. Workspace control orchestrates mounting of the requisite disks and it can also generate blank partitions for images that require extra disk space.

Workspace control connects the deployed VMs to the network via a mechanism that was designed to make the IP assignment process independent of any site DHCP servers while still leveraging this prevalent IP assignment mechanism. It also bootstraps a trusted network for the VMs: the MAC address and the IP address that are chosen for a specific VM are communicated to workspace control by the workspace service. During instantiation, the VM's NICs are each assigned MAC addresses, each connected to a specific bridge port of a Linux bridge in domain 0, and ebtables is configured to recognize the associations and prevent any divergence. The MAC address and IP address association is configured in the DHCP delivery tool which intercepts a VM's boot-up sequence DHCP broadcast, giving the correct IP address to the requests based on the request source's MAC address.

The contextualization information (information allowing the cluster to interpret its context, see Section 6) is currently conveyed by "patching" the deployed image (i.e., mounting the image and copying the information into a well-defined location). Other methods are described in [18].

Workspace control is implemented is a set of lightweight Python programs and shell scripts installed on all the nodes managed by the workspace service. Its main dependencies are a DHCP delivery tool that aids in assigning IP addresses to VMs

and the *ebtables* bridging packet filter package for Linux. The workspace service communicates with workspace-control via an SSH-based protocol.

4 Experimental Evaluation

We evaluated the standup and teardown times for the virtual cluster within our system. The experiments were conducted on 16 nodes of the TeraPort cluster [13], managed by Torque 2.2.1 and Maui Cluster Scheduler 3.2.6-19, and consisting of AMD64 IBM Opteron nodes with 4GB RAM each, connected by gigabit ethernet. The nodes were configured with Xen 3.1.0. During the measurements described here only the workspace pilot jobs were submitted to the nodes, such that they could be executed instantly. We measured creation times for differently-sized virtual clusters composed of single CPU nodes with 1GB of RAM. To isolate the performance specific to our service we assumed that images are already available on the nodes (work on coordinating image transfer and deployment can be found in [19]).

The measurements were taken by storing local timestamps of particular events during cluster create/destroy sequences so we synchronized the clocks on all the machines using NTP. The results shown represent a mean taken over 45 measurements for each cluster size (for N>1 we took the mean over the nodes participating in the iteration). While measuring job startup times with Torque we found that in about 20% of the cases job startups for large N (15 and 16) would be delayed in scheduler queue -- this behavior was found to be specific to the site configuration which was outside of our control; we thus discarded those measurements.

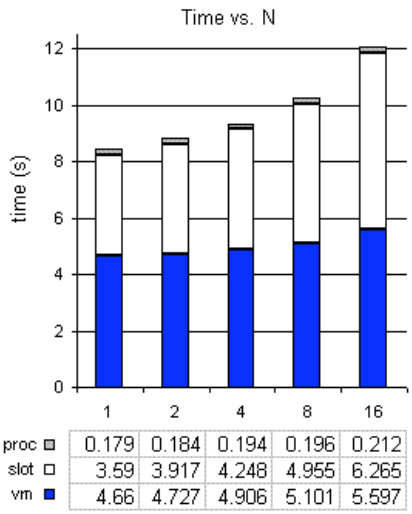

Figure 2: End to end client time

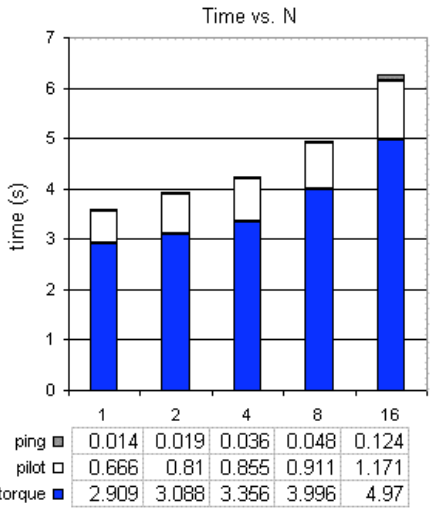

Figure 3: Slot provisioning

We first timed how long it takes to adapt a node for VM deployment using the Xen balloon driver. We used the the xm mem-set command to reduce the memory in domain 0; on average this took 743.6 ms (SD = 100.5). Restoring the memory includes system checks (e.g., for running VMs) and took 1317.2 ms (SD = 96.5). The kill-all

command took 2336.27 ms (SD=236.7). Our trials showed no correlation to the size of adjusted memory and the time of the operation; all numbers were computed as arithmetic mean over 100 trials adjusting 1GB of memory.

Figure 2 shows the time elapsed between the time when a client (located on the same node as the workspace service) submits a request and the time when the client receives notification of request completion. We broke the time into three parts: slot provisioning (*slot*), VM provisioning (*vms*), and client-side processing (*proc*). As can be seen, the time increases slightly as the number of nodes is doubled.

We then examined slot provisioning time in detail (Figure 3). The main component responsible for the time increase is the Torque startup time (*torque*) and is caused by the *pbsdsh* program used by Torque to start jobs of more than one process (in general, this component depends on the LRM used). The time increase in the pilot program (*pilot*) was tracked down to the use of *sudo* for invoking the memory reservation process: the cluster's user accounts are backed by an LDAP database and more (virtually simultaneous) memory calls put extra load on the LDAP server. The notification time (*ping*) increases the load on the workspace service as it is required to receive and respond to increasingly more HTTP notifications.

Next, we looked at the time to create a virtual cluster (Figure 4). The invocation of workspace control mechanisms (*invoke*) is currently implemented with SSH and again with increasing N it puts increasing load on the workspace service (the sudo issue also plays a role). Faster messaging mechanisms that implement collective communication will reduce this time by an order of magnitude or more. The bulk of the time is spent in starting the VMs and connecting them to the network (*create*). The slight decreasing tendency with increasing N is deceptive: it is accounted for by significant timing differences between individual nodes (VMs for small N were running on slow nodes hence the higher mean. The time spent in notifications (*post*) is insignificant compared to the other times.

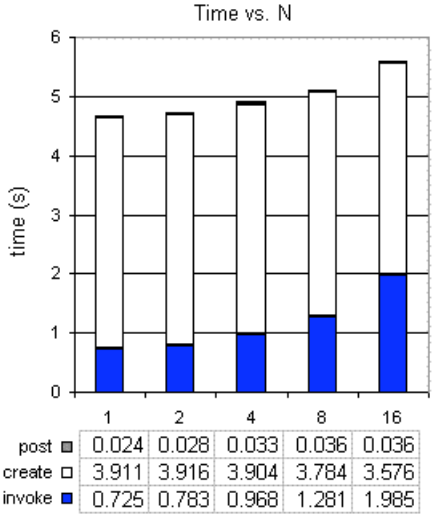

Figure 4: Service mechanisms without LRM and pilot time

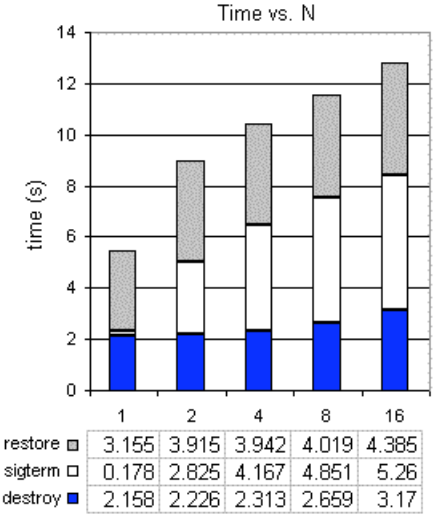

Figure 5: Destruction overview

Figure 5 shows the time elapsed during a typical destruction sequence: a client issues an explicit destroy request. In the figure below *destroy* shows the time it takes the workspace service to terminate all VMs, SIGTERM shows the time it takes for the scheduler to post a SIGTERM signal as a result of qdel request generated by the workspace service, and *restore* shows the pilot restore operation that includes memory adjustment and a substantial idling period. As in the creation times, the LRM is the least scalable component of the time although the use of sudo in the pilot invocation and SSH request processing also contribute to slowdown for larger N to a lesser degree. The use of *pbsdsh* accounts for the sudden jump of the *sigterm* component. When N is greater than one, *pbsdsh* runs on one node, the SIGTERM signal is sent to the local pilot process, and then only after several seconds is the message propagated to the other nodes in the group.

6 Related Work

Multi-scheduling systems have been proposed before. The Condor glide-in mechanism [3] uses a "pilot program" approach similar to the one described here to provision resources which then join a Condor pool. The MyCluster project [4] uses a similar method to create Condor and Sun Grid Engine (SGE) clusters provisioned on top of TeraGrid resources. The Falkon system [6] provisions local resources to deploy a scheduler optimized for handling fine-grained high-througput tasks. All these approaches use LRMs to dynamically provision local resources on which they then overlay a custom scheduling mechanism. Our approach is different in that we deploy VMs over the dynamically provisioned resources; then the provisioned VMs can be further differentiated (e.g. join different scheduling pools). A cluster provisioned with the workspace pilot does not restrict the client's choice of a scheduler (in fact the client need not use a scheduler at all).

Many groups have also explored the integration of LRMs and virtualization. The Dynamic Virtual Clustering (DVC) system [20] integrates the Moab scheduler [21] with Xen to create virtual clusters on a per-job basis so as to provide a unique software environment for a particular application or a consistent software environment across multiple heterogeneous clusters (similar mechanisms are supported in the production version of the Moab scheduler [22]). Fallenbeck et al. [23] proposed Xen-based extensions to the SGE using two VM images (one representing the environment required for parallel and one for serial jobs) to optimize the scheduling functions of the cluster by suspending and resuming those images. All these approaches assume a priori preparation and vetting of images by the cluster administrator and deploy images on a per-job basis (i.e., the modified scheduler still dispatches and manages jobs running inside the VMs). Our approach is different in that we lease out the provisioned VMs to be used via mechanisms independent of the original scheduler, allow clients to request the deployment of arbitrary images and use the contextualization process to adapt them to a particular deployment.

The leasing approach has also been explored by the Shirako project [24], the VioCluster project [25], the Maestro-VS project [26] and the "cluster on the fly" project [27] all explore a leasing-based mode of cluster provisioning. But whereas our ap-

proach in this paper is to provide a leasing environment for VMs within the constraints of an existing scheduling infrastructure, the approaches described above propose new schedulers that could be develop to schedule and deploy VMs.

7 Conclusions

We have described a method that can be used to adapt a job hosting platform to provide a basic VM hosting ability in a non-invasive way. We describe both the implementation of the system and the client's view of provisioning the virtual resources. While this method gives the client limited "terms of service" (constrained by the policies implemented by the batch scheduler), it also provides a simple way for existing resource providers to experiment with VM hosting.

Our evaluation shows that, assuming image availability, virtual clusters can be provisioned reasonably cheaply and scalably using this approach: a 16 node cluster can be provisioned in 12 seconds including VM boot time (as compared to 8 seconds for a cluster of 1). In our experiments, the least scalable component of provisioning proved to be the LRM which took half the time of overall end-to-end deployment.

Acknowledgements

This work was supported by NSF SDCI award #0721867 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 Greg Cross and Ti Leggett for discussion of the workspace pilot implementation.

References

1. Chase, J., L. Grit, D. Irwin, J. Moore, and S. Sprenkle, *Dynamic Virtual Clusters in a Grid Site Manager*. HPDC-12, 2003.
2. *Amazon Elastic Compute Cloud (EC2)*: www.amazon.com/ec2.
3. Frey, J., T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, *Condor-G: A Computation Management Agent for Multi-Institutional Grids.* Cluster Computing, 2002. **5**(3): p. 237-246.
4. Walker, E., J. Gardner, V. Litvin, and E. Turner, *Creating Personal Adaptive Clusters for Managing Scientific Jobs in a Distributed Computing Environment.* CLADE, 2006.
5. Nilsson, P., *Experience from a pilot based system for ATLAS. CHEP 2007.*
6. Raicu, I., Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde, *Falkon: a Fast and Light-weight tasK executiON farmework.* SuperComputing, 2007.
7. Agarwal, A., R. Desmarais, I. Gable, A. Norton, R. Sobie, and D. Vanderster, *Evaluation of Virtual Machines for HEP Grids*. CHEP 2006.

8. Keahey, K., T. Freeman, J. Lauret, and D. Olson. *Virtual Workspaces for Scientific Applications*. in *SciDAC Conference*. 2007.
9. Freeman, T., K. Keahey, I. Foster, A. Rana, B. Sotomayor, and F. Wuerthwein, *Division of Labor: Tools for Growth and Scalability of the Grids.* ICSOC 2006.
10. Keahey, K., I. Foster, T. Freeman, and X. Zhang, *Virtual Workspaces: Achieving Quality of Service and Quality of Life in the Grid.* Scientific Programming Journal, 2005.
11. *Torque*: http://www.clusterresources.com/pages/products/torque-resource-manager.php.
12. *Sun Grid Engine:* http://gridengine.sunsource.net.
13. *The TeraPort Cluster*: http://www.ci.uchicago.edu/research/detail_teraport.php.
14. Czajkowski, K., D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe, *The WS-Resource Framework*. 2004: http://www.globus.org/wsrf.
15. *Virtual Workspaces*: http://workspace.globus.org.
16. Barham, P., B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebar, I. Pratt, and A. Warfield. *Xen and the Art of Virtualization*. in *ACM Symposium on Operating Systems Principles (SOSP)*.
17. Freeman, T., K. Keahey, B. Sotomayor, X. Zhang, I. Foster, and D. Scheftner, *Virtual Clusters for Grid Communities.* CCGrid, 2006.
18. Bradshaw, R., N. Desai, T. Freeman, and K. Keahey. *A Scalable Approach to Deploying and Managing Virtual Appliances*. in *TeraGrid 2007*
19. Sotomayor, B., K. Keahey, and I. Foster. *Overhead Matters: A Model for Virtual Resource Management*. in *1st International Workshop on Virtualization Technology in Distributed Computing (VTDC)*. 2006.
20. Emeneker, W., D. Jackson, J. Butikofer, and D. Stanzione, *Dynamic Virtual Clustering with Xen and Moab.* Workshop on Xen in HPC Cluster and Grid Computing Environments (XHPC), 2006.
21. *The MOAB Workload Manager*: http://www.clusterresources.com/pages/products/moab-cluster-suite/workload-manager.php.
22. *MOAB Administrator's Guide: Virtualization and Resource Provisioning*: http://www.clusterresources.com/products/mwm/docs/5.6resourceprovisioning.shtml.
23. Fallenbeck, N., H. Picht, M. Smith, and B. Freisleben, *Xen and the Art of Cluster Scheduling.* VTDC, 2006.
24. Irwin, D., J. Chase, L. Grit, A. Yunerefendi, D. Decker, and K. Yocum, *Sharing Networked Resources with Brokered Leases.* USENIX Technical Conference, 2006.
25. Ruth, P., P. McGachey, and D. Xu, *VioCluster: Virtualization for Dynamic Computational Domains.* IEEE Conference on Cluster Computing, 2005.
26. Kiyanclar, N., G.A. Koenig, and W. Yurcik, *Maestro-VS: A Paravirtualized Execution Environment for Secure On-Demand Cluster Computing.* CCGrid, 2006.
27. Nishimura, H., N. Maruyama, and S. Matsuoka, *Virtual Clusers on the Fly -- Fast, Scalable and Flexible Installation.* CCGrid, 2007.