

# Flying Low: Simple Leases with Workspace Pilot

Timothy Freeman, Katarzyna Keahey

Computation Institute  
University of Chicago  
{t.freeman, keahey}@uchicago.edu

**Abstract.** As virtual machines (VMs) are used for a wider range of applications, we encounter the need to integrate virtual machine provisioning models into the current site resource management infrastructure as seamlessly as possible. To address such requirements, we describe an approach to VM management that uses the ideas of multi-level scheduling to integrate VM provisioning into existing job schedulers. We further describe how VMs provisioned in this way can be turned into useful virtual clusters integrated into the surrounding infrastructure and community.

## 1 Introduction

Virtual machines (VMs) have many features that make them attractive for running scientific applications, such as security and isolation, the ability to deploy customized software environments, fine-grained resource control, and site-independence [1, 2]. In particular, using VMs in the grid setting has the potential to allow application groups to lease out resources from grid providers without having to adapt these resources to site-specific configurations. Such short-term leases, obtained via the vehicle of VMs, can seamlessly and securely overlay the configuration required by the various virtual organizations (VOs) thus making grid computing easier for both consumers and providers [3]. However, despite its attractiveness to both groups, this model of resource provisioning has not so far seen widespread adoption due to a relatively high barrier involved in adapting a site infrastructure to the needs of VM deployment.

The existing grid sites typically manage their resources using batch schedulers and local resource managers (LRMs) such as Torque [4] or SGE [5]: users submit batch job requests to those schedulers and their requests are executed as resources become available. To easily integrate VM provisioning within those sites we need a mechanism that would easily combine with existing scheduler installations (i.e., without requiring modifications to their source code) and operate within familiar metaphors to provide dynamic leases to consumers. It is essential that such mechanism can flexibly combine job and VM scheduling and allow for both modes of operation within a site.

In this paper, we describe how multi-level scheduling can be used to adapt the current LRMs for VM hosting. We describe the design and the implementation of resource provisioning as well as the logistics of VM deployment and integration into the networking infrastructure. Further, we describe how a grid client can dynamically provision useful virtual clusters using our infrastructure. The system described here,

codenamed “workspace pilot”, has been developed for the TeraPort cluster at the University of Chicago [6] using Torque as the local LRM; we present the evaluation of virtual cluster deployment on that system.

## 2 Approach

We use a multi-level scheduling approach, similar to that employed by Condor [7] glide-ins, to enable resource leasing with VMs. This approach relies on submitting a job request to the local scheduler with a specified resource allocation request. 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.

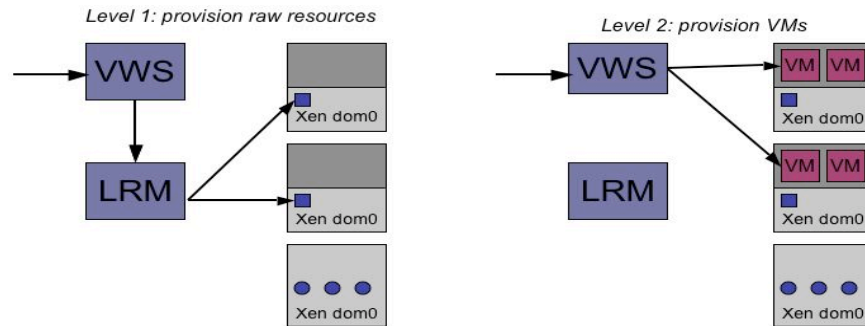
### 2.1 The Workspace Service

The workspace service [8] allows an authorized remote client to provision workspaces (implemented as virtual machines) on physical resources under its control. 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). To deploy a VM, a client makes a request to deploy  $n$  VMs each associated with the same image and resource allocation (describing the duration of the lease, as well as its shape: memory, CPU, etc. available to the VM) [3, 9]. The workspace service allows the client to access information related to workspaces and manage them individually or as a group. The client can also subscribe for notifications of events related to the workspace lifecycle (e.g., it can be notified when a workspace is deployed). To make it possible for clients to request flexibly configured heterogeneous virtual clusters, we implemented the The Ensemble Service which allows the client to group together several heterogeneous workspaces (using different images or resource allocations) to be managed together. Workspaces may be added to the ensemble before deployment only. While the services are responsible for processing client requests, the workspace backend infrastructure deploys the corresponding VMs on a static set of physical machines.

### 2.1 Two-Level Provisioning

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. We assume 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 [10] 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 resource*). 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 scheduled by the LRM adjusts the memory, 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 that are to be scheduled in that resource slot. Then, the pilot notifies the workspace service that the resource slot has been obtained. When the resource slot terminates (either by request from the workspace service, because it has expired, or the pilot receives a catchable termination signal from the LRM) the pilot gracefully terminates the slot collaborating with the workspace service on cleanup actions.

The provisioning of virtual machines takes place 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 available 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.

The Workspace Service may implement a variety of policies when requesting resources via the pilot and mapping VMs onto those resources, e.g. physical resources may be provisioned pro-actively or in direct response to a request, they may be over-provisioned (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 re-

quest. Further, the default policy always translates the requested resources into the smallest resource slot into which they fit such that it can be provisioned by the LRM, mapping one VM per node.

## 2.2 Deploying and Configuring Virtual Clusters

Most scientific computations are performed on homogenous or heterogeneous clusters rather than collections of independent resources. A cluster is typically characterized by a common context between the nodes: e.g., they are managed by a scheduler or use common networked storage. Further, in order to be useful to a community or an individual, a cluster may have to connect to the individual's or community's security and information structures (e.g., be configured with the right public keys or host certificates). These deployment-specific adaptations are typically applied to VMs manually. However, especially for deployments that are dynamic and short-term, it is important that the process through which the deployed cluster becomes a useful entity is fully automated. In this section we describe this process and describe the way to flexibly configure virtual clusters out of VMs.

We create a common context between the deployed VMs and integrate them into a larger deployment context of a site, a community or a grid via a process called *contextualization* described in detail in [11]. Contextualization relies on appliance/VM provider to export a *contextualization template*: a description of information required to integrate a VM into its deployment context (e.g., the need to know the IP addresses of all VMs forming a cluster). The appliance provider also describes actions that consume this information and adapt the configuration of applications to integrate it. On VM deployment, contextualization information is provided by the VM deployer; once the VM boots, the actions are performed and the VM is integrated into the context.

We rely on this process, as well as workspace features described above to allow grid clients to dynamically deploy flexibly configured clusters. For example, in order to deploy an OSG cluster consisting of a compute element (CE), a storage element (SE) and  $n$  worker nodes [3], a client performs the following actions. The client first requests the creation a workspace consisting of  $n$  worker nodes, and workspaces representing the SE and CE respectively. 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. All these workspaces are added to a workspace ensemble. Once the client is satisfied that the virtual cluster has all the required elements, the ensemble is deployed. On deployment, contextualization integrates the networking information into the job and storage management applications and results in the deployment of a fully functional cluster.

## 3 Implementation

We implemented the approach described above by developing 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 component is implemented by the workspace service and consists of control and monitoring interfaces to the LRM (e.g., *qsub* and *qdel* commands for Torque). The second component is implemented within the pilot process and consists of signal handlers. 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 implementation of the signal handler is described below.

### 3.2 The Workspace Pilot

The actions required to adapt a job platform for VM hosting depends on how much these two platforms have in common. 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 account the pilot runs in to have 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 [3]. The absolute minimum required by domain 0 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, after the expected duration of the slot has expired the pilot executes the “release slot” operation that will completely undo the effects of the “reserve slot”.

Occasionally the pilot program may receive a catchable (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 offers a “grace period” before 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 Deployment: Workspace Control Implementation

Workspace control 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 Linux “ebtables” bridging packet filter package. The workspace service communicates with workspace-control via an SSH-based protocol.

The primary function of workspace control is to start, stop and pause VMs based on commands received from the workspace service. To assist with VM deployment, workspace control can trigger VM propagation of VM images from a location within the site to the node on which it executes. Further, since the workspace service allows clients to request image reconstruction from disk partitions cached on the site (to a limited extent), it also orchestrates mounting of the requisite partitions. Finally, it can generate blank partitions for images that require extra blank 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.

## 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 [6], 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.

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 configuration which was outside of our control; we thus discarded those measurements.

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 (PROCESSING). We first 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 will vary depending on the properties of 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 pilot 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.

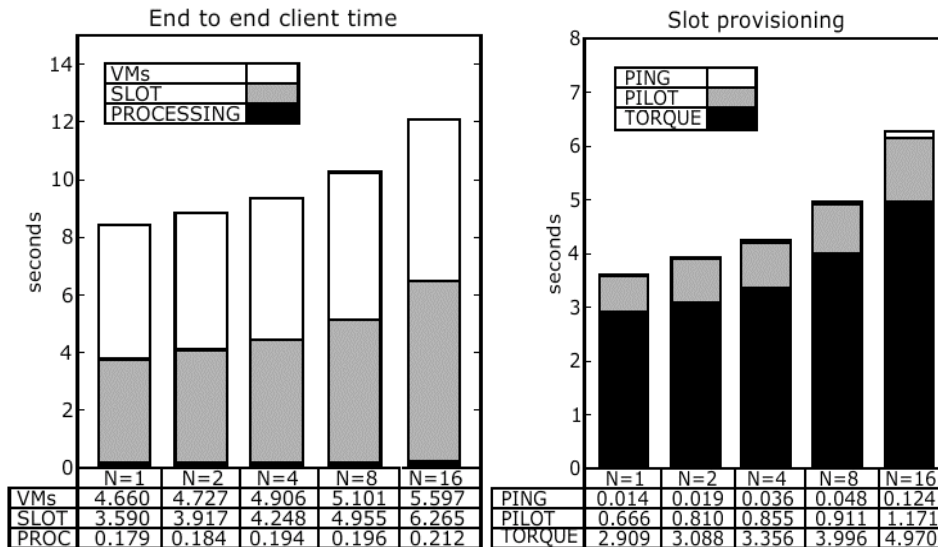


Figure 2: End to end client time

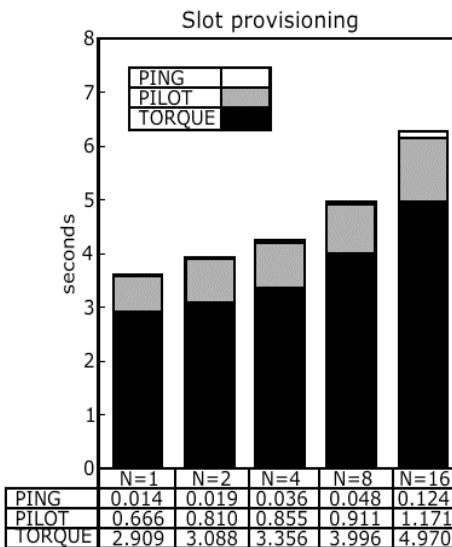


Figure 3: Slot provisioning: (1) Submission to Torque, (2) Pilot startup, and (3) notification ping

We then examined 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). Messaging mechanisms including group and broadcast functionality would reduce this time. 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); we are still investigating these differences. The time spent in notifications (POST) is insignificant compared to the other times.

Service mechanisms without LRM and pilot

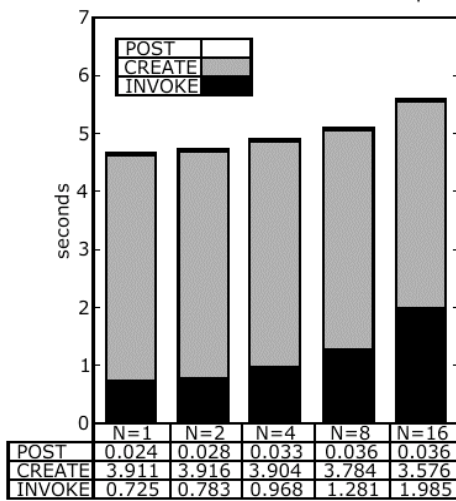


Figure 4: Service mechanisms without LRM and pilot time

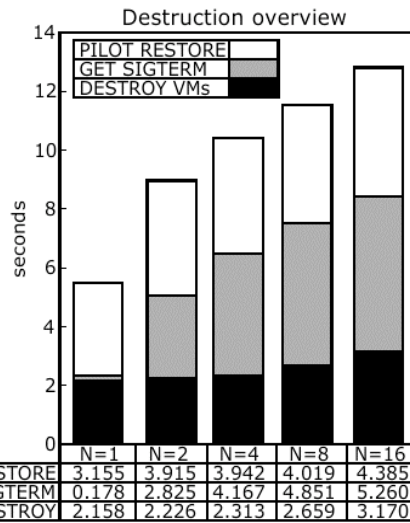


Figure 5: Destruction overview

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 interrupt 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 (*pbsdsh* use accounts for SIGTERM behavior) although the use of sudo in the pilot invocation and SSH request processing also contribute to slowdown for larger N albeit in a much lesser degree.

## 5 Related Work

Multi-scheduling systems have been proposed before. The Condor glide-in mechanism [7] uses a “pilot program” approach similar to the one described here to provision resources which then join a Condor pool. The MyCluster project [12] uses a similar method to create Condor and Sun Grid Engine (SGE) clusters provisioned on

top of TeraGrid resources. The Falkon system [13] provisions local resources to deploy a scheduler optimized for handling fine-grained high-throughput 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 [14] integrates the Moab scheduler [15] 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 [16]). Fallenbeck et al. [17] 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 [18], the Vio-Cluster project [19], the Maestro-VS project [20] and the "cluster on the fly" project [21] all explore a leasing-based mode of cluster provisioning. But whereas our approach 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 developed to schedule and deploy VMs.

## 6 Conclusions

We have described a method that can be used to adapt a job hosting platform used on many sites today 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 also took roughly half the time of overall end-to-end deployment.

## References

1. Agarwal, A., R. Desmarais, I. Gable, A. Norton, R. Sobie, and D. Vanderster, Evaluation of Virtual Machines for HEP Grids. CHEP'06, 2006.
2. Keahey, K., T. Freeman, J. Lauret, and D. Olson. Virtual Workspaces for Scientific Applications. in SciDAC Conference. 2007. Boston, MA.
3. Freeman, T., K. Keahey, I. Foster, A. Rana, B. Sotomayor, and F. Wuerthwein, Division of Labor: Tools for Growth and Scalability of the Grids. ICSSOC, 2006.
4. Torque: <http://www.clusterresources.com/pages/products.php>
5. Sun Grid Engine Web Site. <http://www.sun.com/software/gridware/>
6. The TeraPort Cluster: [http://www.ci.uchicago.edu/research/detail\\_teraport.php](http://www.ci.uchicago.edu/research/detail_teraport.php).
7. Frey, J., T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. in 10th IEEE International Symposium on High Performance Distributed Computing. 2001
8. 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.
9. Virtual Workspaces Web Page: <http://workspace.globus.org>.
10. 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).
11. Bradshaw, R., N. Desai, T. Freeman, and K. Keahey. A Scalable Approach to Deploying and Managing Virtual Appliances. in TeraGrid 2007
12. Walker, E., J. Gardner, V. Litvin, and E. Turner, Creating Personal Adaptive Clusters for Managing Scientific Jobs in a Distributed Computing Environment. CLADE, 2006.
13. Raicu, I., Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde, Falcon: a Fast and Light-weight task execution framework. SuperComputing, 2007.
14. 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.
15. MOAB: <http://www.clusterresources.com/pages/products.php>
16. MOAB Administrator's Guide: Virtualization and Resource Provisioning: <http://www.clusterresources.com/products/mwm/docs/5.6resourceprovisioning.shtml>.
17. Fallenbeck, N., H. Picht, M. Smith, and B. Freisleben, Xen and the Art of Cluster Scheduling. VTDC, 2006.
18. Irwin, D., J. Chase, L. Grit, A. Yunerefendi, D. Decker, and K. Yocum, Sharing Networked Resources with Brokered Leases. USENIX Technical Conference, 2006.
19. Ruth, P., P. McGachey, and D. Xu, VioCluster: Virtualization for Dynamic Computational Domains. IEEE International Conference on Cluster Computing, 2005.
20. Kiyancilar, N., G.A. Koenig, and W. Yurcik, Maestro-VS: A Paravirtualized Execution Environment for Secure On-Demand Cluster Computing. CCGrid, 2006.
21. Nishimura, H., N. Maruyama, and S. Matsuoka, Virtual Clusters on the Fly -- Fast, Scalable and Flexible Installation. CCGrid, 2007.