# Virtual Workspaces in the Grid

Katarzyna Keahey[1], Ian Foster[1,2], Timothy Freeman[2],
Xuehai Zhang[2], and Daniel Galron[3]

[1] Argonne National Laboratory
{Keahey,foster}@mcs.anl.gov
[2] The University of Chicago
{tfreeman,hai}@cs.uchicago.edu
[3] The Ohio State University
galron@cis.ohio-state.edu

**Abstract.** Despite significant progress in the development of Grid infrastructure, the provisioning of a customized and controllable remote execution environment remains an open issue. This paper introduces the concept of a virtual workspace, a configurable execution environment that can be created and managed as a first-class entity to reflect client requirements. Such workspaces can be dynamically deployed on a variety of resources decoupling the notion of environment and resource. We show how virtual workspaces fit into the Grid architecture, present an example implementation using virtual machines, and discuss our initial experiences using this system in practice and with applications.

## 1 Introduction

While significant progress has been achieved with the deployment of Grid-based applications, the preparation of a remote execution environment remains an issue. One of the reasons is that while Grids offer access to diverse software environments, an application typically requires a very specific, customized environment. As a consequence of variations in operating system, middleware versions, library environments, and file system layouts a user's application may in practice be able to use only a small fraction of the resources potentially available on the Grid. The second issue is the need to provide reliable isolation and dynamic, fine-grain control of shared resources to ensure enforcement of policies and thus provide incentive for wider sharing. The development of Grid protocols [1, 2] provides uniform ways to manage Grid entities that could represent such environments. It now remains to find ways to describe represent and implement them.

In this paper, we present the concept of a virtual workspace, which allows a Grid client to define an environment in terms of its requirements (such as resource requirements or software configuration), manage it, and then deploy the environment in the Grid. Workspaces defined in this way can be implemented in a variety of ways such as for example dynamically creating Unix accounts and using system as well as software configuration tools to enforce the required properties. Here, we focus on a particularly promising implementation of virtual workspaces: virtual machines (VMs). The use of virtual machines in Grid computing has been proposed before [3, 4]. In addition to outstanding isolation properties, VMs can provide fine-grained enforcement; and by their very nature—virtualization of the underlying hardware—

J.C. Cunha and P.D. Medeiros (Eds.): Euro-Par 2005, LNCS 3648, pp. 421–431, 2005.

they enable instantiation of a new, independently configured guest environment on a host resource. They can be rapidly suspended and their state serialized, and thus easily migrated to remote resources. Moreover, as a result of recent progress in virtual machine technology, these advantages no longer come at a performance cost to either the application or the hosting resource: systems such as Xen [5] demonstrate that they can be used with little or no performance degradation.

In the rest of the paper, we describe how virtual workspaces fit into the Grid architecture, present a prototype of this architecture based on the Globus Toolkit (GT) and experiment with two workspace implementations using Xen [5] and the VMware Workstation [6]. We describe our initial experiences with integrating VMs into the Grid infrastructure, and we present preliminary results of testing our prototype system with a bioinformatics application suite.

## 2 Virtual Workspaces

Interactions in present-day Grids focus on mapping jobs to resources, often with the assumption that an execution environment with suitable configuration and enforcement characteristics will be provided by means independent of the Grid infrastructure. Although such an assumption is true for closely knit groups of Grid users, it is not justified when applications or users with drastically different requirements and rights are trying to use the same resources. Recognizing this fact, we define the concept of a *virtual workspace* that can be automatically deployed on resources and provide a required execution environment. Thus, jobs can be mapped to workspaces, and workspaces can be mapped to actual resources in the Grid.

A virtual workspace (VW) is a definition of an execution environment in terms of its hardware requirements, software configuration, isolation properties, and other salient characteristics. The intent of defining a workspace is to capture the requirements for an execution environment in the Grid and then use automated tools in order to find, configure, and provide an environment best matching those requirements. We could, for example, use agreement-based tools to negotiate contracts defining workspaces and the use of actual resources and then negotiate a binding between them following the models described in [2, 7, 8]. Depending on the requirements, such contracts could be fulfilled by simply dynamically creating and configuring user accounts as in [9], by using pre-configured virtual machines or other sandboxing and virtualization technologies.

### 2.1 Virtual Workspace Descriptions

To describe workspaces, we use an XML Schema, which captures generic properties of every workspace, as well as properties subject to specific workspace definition or workspace implementation-specific properties.

The XML example below shows a description of a workspace. It contains the workspace category type that describes what mechanisms are used to create the workspace: we currently support implementations based on different types of virtual machines and dynamic accounts [10]. Currently, the implementation type is used to define the isolation model as well as provide a clue to services processing the workspaces to provide implementation-specific processing. Workspace state can be one of running (a workspace deployed on a resource), shutdown (for example, a “cold” VM

image containing no running processes), paused (for example, a "hot" VM image), and corrupted (a workspace that cannot be deployed because of internal inconsistencies). In addition, the generic part of workspace description contains a reference that can be used to check the status of the workspace. The reference embeds information about the workspace owner's distinguished name. Other properties contained in the generic part of the schema include three time-related elements: creationTime, lifeCycle, and lastModified. CreationTime records the time when the workspace was first instantiated. LifeCycle indicates how long the workspace is available for use. LastModified keeps the information about when the properties of the workspace were modified last.

Further definition contains a description of different workspace aspects, such as required hardware, networking configuration, required software installations and workspace capability. Description of the "virtual resource" that represents the hardware requirements of the execution environment contains elements such as the RAM size, disk size, disk type, and accessing mode, as well as devices such as virtual CD-ROM drives. A network specification contains the description of a network connection and how to establish it (such as the method to obtain an IP address). Software descriptions contain information about the operating system (e.g., kernel version, distribution type), library signature, and programs installed. Workspace capability describes what can be done with the workspace: for example, a workspace may be configured to run a program on startup or have some programs (in particular, hosting programs) already running and be able to service requests on specific ports.

```
<xs:simpleType name="categoryType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="dynamic account"/>
    <xs:enumeration value="vm"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="stateType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="shutdown"/>
    <xs:enumeration value="paused"/>
    <xs:enumeration value="running"/>
    <xs:enumeration value="corrupted"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="virtualWorkspace">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="category" type="categoryType" default="vm"/>
      <xs:element name="state" type="stateType" default="shutdown"/>
      <xs:element name="EPR" type="wsa:EndpointReferenceType"/>
      <xs:element name="creationTime" type="xs:dateTime" minOccurs="0"/>
      <xs:element name="lifeCycle" type="xs:integer" minOccurs="0"/>
      <xs:element name="lastModified" type="xs:dateTime" minOccurs="0"/>

      <xs:element ref="hw:hardware" minOccurs="0"/>
      <xs:element ref="net:network" minOccurs="0"/>
      <xs:element ref="sw:software" minOccurs="0"/>
      <xs:element ref="cap:capability" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Wherever applicable, workspace properties are described as a set of possible values (e.g., RAM size with min and max requirements) rather than one value. The intent is to leave open the largest possible set of mappings of workspaces to real resources. The schemas are extensible to reflect the capabilities of different implementations. For example, we use a VM-based workspace category type, extended from the generic category type, to describe the category information of a workspace implemented with virtual machines. Besides the inherited category name, it contains extra properties such as the type of virtual machine monitor (VMM), which are specific for VM-based workspaces only.

Based on the descriptions defined in the schema, workspaces can be selected, cloned, or refined. Cloning a workspace, for example, involves creating a new name (as encoded within the reference element), a new resource and copying the description metadata.

### 2.2 Virtual Workspaces as Virtual Machines

We have surveyed candidate technologies for workspace implementations [4] and identified two especially promising ones: configurable dynamic accounts [10, 11] and virtual machines. Although dynamic accounts present an interesting implementation option (especially when used with enforcement tools such as quota and software configuration tools such as Pacman [12]), we pursue this work elsewhere [9]. Our focus in this paper is on a virtual machine implementation of workspaces in view of their outstanding isolation and serialization properties.

A virtual machine [13] provides a virtualization of the underlying physical host machine. Software running on the host, called virtual machine monitor (VMM) or hypervisor, is responsible for supporting the perception of multiple isolated machines by intercepting and emulating privileged instructions issued by the guest machines. A VMM typically also provides an interface allowing a client to start, pause or stop multiple guests. A VM representation contains a full image of a VM's RAM, disk, and other devices, allowing its state to be fully serialized, preserved, and restored at a later date. Recent exploration of paravirtualization techniques [5] has led to substantial performance improvements in virtualization technologies, making virtual machines an attractive option for Grid computing.

The serialization properties of VMs create the potential for effortless configuration of execution environments (for example, allowing a user to configure VMs with software required by a given community and clone them). Their isolation from the host machine provides a way of running a different configuration from that of the VM host and allows guest VMs and resource owners to take advantage of enhanced security properties of VMs. Further, those abilities combined provide potential for migration. For these reasons, VMs provide an excellent implementation option for workspaces: the configuration of a VM image can reflect a workspace's software requirements while the VMM can ensure the enforcement of hardware properties.

## 3 Integrating Virtual Workspaces into the Grid Architecture

Virtual workspaces refine the execution environment layer in Grid architecture: rather than mapping jobs directly onto hardware resources as in [14], we map jobs to preconfigured workspaces which can then be mapped to Grid resources. Since a work-

space may exist beyond its deployment, and may in fact may be deployed many times on different resources during its lifetime, we introduce two new services: VW Repository which provides a management interface to workspaces, and VW Manager which orchestrates their deployment. The figure below illustrates how workspaces work within the Grid infrastructure:

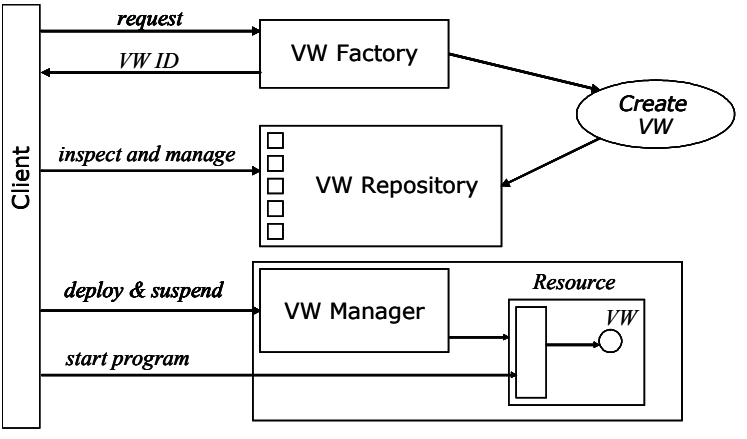


**Fig. 1.** Grid Interactions with Virtual Workspaces

### 3.1 Grid Interactions

In order to create a workspace instance, a Grid client contacts the VW Factory with a workspace description presented in Section 2.1. A negotiation process may take place to ensure that the workspace is created in a policy controlled way. The newly created workspace is registered with a VW Repository, which provides a Grid service interface allowing for inspection and management of workspaces and keeps track of resources implementing workspaces such as virtual machine images. As a result of creation the client is returned a WSRF end-point reference (EPR) to the workspace. We leverage the abstraction of a Grid service resource [1] to enable inspection and management of properties such as termination time.

To deploy a VW on a specific resource, a client contacts a VW Manager Grid service on that resource and presents it the workspace's EPR. The VW Manager allows a client to deploy/undeploy, start/stop, and (in our VM-based implementation) also pause/unpause workspaces. Deploying a workspace simply means staging all components of the workspace (such as a VM image) to the resource so that they are available to the VW Manager on that resource. Currently, "checking out" workspaces from the repository puts a lock on them, since their state might change during use. Similarly, "undeploy" releases the hold on workspace resources and the corresponding lock. After the workspaces are staged, they can be started (i.e., become available for computation). Once a VW becomes ready for execution, a program can be started by using Grid infrastructure mechanisms (e.g., Globus Resource Allocation Manager, or GRAM) or by using other methods such as preconfigured program startup or a continuation of a previous execution. The VW Manager can also stop, pause ("freeze" an ongoing computation), or undeploy a stopped or paused workspace (stage it back to the repository for example).

In a complete picture of interactions, a community broker would negotiate reservations or agreements for the use of specific resource allocations on a resource. The workspace agreements would be matched against those allocations and create binding agreements allowing the deployment of a specific workspace on a selected resource much as described in SNAP [7]. Such agreements could then be renegotiated and the workspaces migrated, as need arises.

### 3.2 Implementation

Our current prototype uses both Xen (version 2.01) and VMware (Workstation, version 4.5) VMs to implement virtual workspaces. The workspaces use IP addresses from a pre-reserved pool for networking. The Grid services and infrastructure described above were implemented by using GT4 (alpha version 3.9.4). In order to be capable of deploying workspaces of a specific type, a host machine has to run a virtual machine monitor of that type as well as the VW Manager service. At this point we assume that no VM image will leave the trusted environment of a specific site; that is, we do not yet introduce mechanisms to protect integrity and privacy of images themselves, and we assume one repository per site.

New workspaces are created by cloning existing images configured with the same requirements and stored in the repository image pool, as in [15]. Each workspace is configured with a certificate and a private key to authenticate itself to clients. At this stage, we do not support negotiation for workspace creation; a workspace request is either accepted or rejected based on existing policies.

The VW Manager interfaces with VMM running on a particular resource to implement to stage/unstage, start/stop, and pause/unpause operations: in Xen via an HTTP control interface and on VMware Workstation via GUI scripting. To stage a workspace, the VW Manager transfers the workspace data (including description metadata and the implementation-specific image) from the VM Repository to the host node by using GridFTP [16]. Once the workspace data transfer is complete, the VW Manager waits for the client to start the workspace, which includes creating a workspace resource, loading the VM image into memory, and booting the VM. At boot time the VM may initiate preconfigured operations such as obtaining its network address and starting programs (including the GT hosting environment). Once this step is completed, a VM can advertise the hosted Grid services such as GRAM for clients to invoke.

Although in the current prototype we do not address the issue of ensuring privacy and integrity of workspace representations (VM images), we do support standard Grid authentication and authorization mechanisms. Running workspaces and Grid clients mutually authenticate by using the Grid Security Infrastructure GSI [17]. Our infrastructure also accepts VOMS certificates [18] and is capable of extracting VOMS attributes for authorization. Authorization of workspace creation, deployment, and management is configured via access control lists based on the distinguished name and attributes of Grid entities.

## 4 Experiences with Virtual Workspaces

The performance impact of virtual machines on applications has been shown to be small (typically under 5% of slowdown) for different application classes [5, 19, 20].

In our preliminary evaluation, we explored the performance impact of different ways of using VMs as part of Grid infrastructure. We also conducted a preliminary evaluation of VM usage with applications.

To explore the best ways of using VMs within the Grid infrastructure, we timed the process of starting up a workspace and running a program in it on a remote node in different startup configurations. We repeated those experiments for both of our workspace implementations (Xen and VMware Workstation) and compared them with the time of job startup through a call to GT4 GRAM. In all the experiments we assume that the necessary data had already been staged to the node (i.e., the executable and input data for GRAM and a VM image in the case of workspace deployment).

All experiments were run on a dual 2.2 GHz Xeon server configured to run single-CPU guest VMs. The same VM image configuration (Debian Sarge) was used for both Xen and VMware, and the same test application was used for all experiments. Time was measured on the server side only, by using wall clock time.

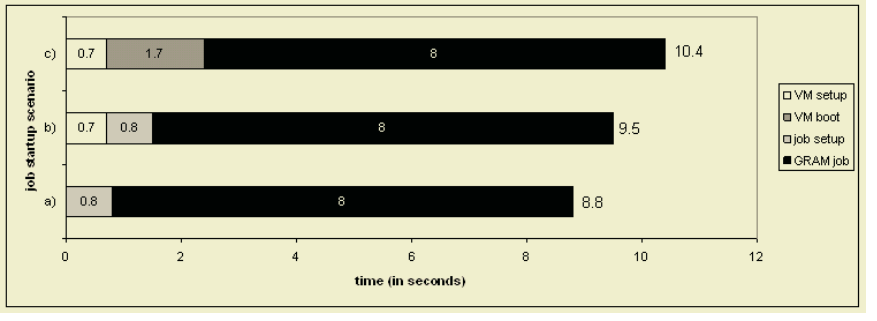


**Fig. 2.** Xen-based VWs versus job invocation using GRAM

Figure 2(a) shows time elapsed from the moment GRAM received a job startup request till the end of the job: as per GRAM default the job is run in a user account. Figures 2(b-c) show the combined time of deploying a workspace implemented as a Xen VM and starting a job in it under two different scenarios. In 2(b) we take advantage of the serialization property of VMs: we prepare a "hot image" (a paused image) with a hosting environment already started up. A call to GRAM is placed as soon as the environment is available. In 2(c), instead of configuring the workspace to start up a GT4 hosting environment, we start the job directly. Unless the VM is partially configured at boot time, this scenario could also be optimized by pausing a booted VM.

For comparison, we also ran a similar test on VMware Workstation, which is hard to time given its non-programmatic, opaque interface (VMware ESX/GSX tools provide more efficient and direct interfaces). The results are summarized in Figure 3; a significant amount of time is spent in a controller adapter to the Workstation version.

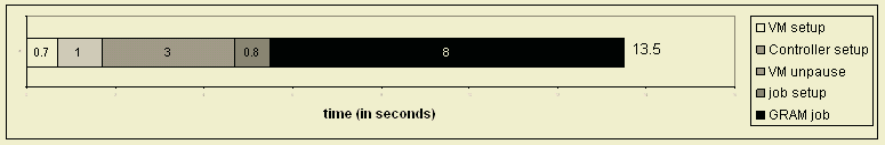


**Fig. 3.** VM deployment time using VMware workstation

Our preliminary experiments show that a Xen VM could be effective for a range of configurations with addition to job deployment time that can easily be absorbed by latency in the Grid environment. Startup costs of deploying a VM image are comparable to starting a job; the trade-off is that while GRAM implements a flexible job deployment strategy the VM is already pre-configured with the job to be started. On the other hand, the advantages gained by using VMs are significant. Deploying hot images ready to process requests offers the fastest solution (note that this method could be used to eliminate application initialization time as well). However, this may not always be possible; in such cases using a VM preconfigured to start a specific job is also an efficient alternative (currently no credential is delegated to such a job; the startup is based on authorization of the client that submits the VM).

To obtain a preliminary assessment of the usefulness of this infrastructure to Grid applications, we experimented with the EMBOSS [21] suite for bioinformatics applications. The most important effect was facilitating deployment: while the EMBOSS installation took roughly 45 minutes, starting a preconfigured VMware Workstation workspace took on average 6 minutes and 23 seconds (including staging), and the process itself eliminated installation errors (as per our results above, Xen could be used for even better results). Another noticeable consequence was a more flexible use of resources, especially in a heterogeneous resource environment: we no longer had to require a homogenous resource base.

While the determination is application specific, our results suggest there are situations where using virtual machine workspace implementations would not be useful. Jobs that require 100% resource utilization, do not require specific environments, or that each run for less time than the time required to stage, boot or unpause a virtual machine would not be good candidates for the infrastructure.

## 5 Related Work

Due to its potential, the use of virtual machines in Grid computing is attracting increasing attention [3]. The In-Vigo project [15, 22] made substantial progress in this direction while the Virtuoso [23] and VIOLIN [24] projects explored networking issues arising from use of VMs in this setting. Our approach differs in that it focuses on technology-neutral virtual workspaces, using virtual machines as only one of their potential implementations. Further, recognizing that a workspace may be deployed many times we distinguish between workspace creation and its deployment introducing a different architecture for its support.

Attempts to capture requirements of an execution environment to some extent and automate their deployment have also been made before: for example, the virtual appliance project [25] uses virtual machines configured based on descriptions in a configuration language to ease administrative burden, and the Cluster on Demand (COD) project [26] allows a user to choose from a database of configurations to configure a partition of a cluster. We differ from these projects by our focus on Grid computing and technology-independent approach.

Finally, the Xenoserver project [27] is building an infrastructure for wide-area distributed computing based on virtual machines similar to our long-term goals. Here, we differ by building within the established framework of Grid computing rather than providing new infrastructure.

## 6 Conclusions and Future Work

We have described the abstraction of a virtual workspace, a customizable execution environment capable of being deployed on a variety of platforms in the Grid. Workspaces are defined in terms of client requirements, such as software and hardware requirements, and are implemented in terms of technologies providing an isolated execution environment, quality of service at the granularity of a workspace (as opposed to a single process), customized software installation, and, in the case of VMs, execution serialization and migration.

We showed how workspaces can be integrated into the existing Grid infrastructure. The integration entails relatively small changes, but introduces substantial flexibility of use. The use of Grid protocols allows us to fully leverage this flexibility and create, deploy, and shut down workspaces dynamically based on policy-driven provisioning decisions. By virtue of their properties, workspaces are a promising vehicle for implementing policy-driven Grid usage.

To evaluate the feasibility of our workspace implementations, we compared the performance of GT4 GRAM, a widely used job startup service for Grid applications, to the process of starting up workspaces implemented as virtual machines in a variety of scenarios. In conjunction with the small performance impact on applications [5], our results show workspaces to be a promising abstraction for Grid computing. In addition, our experiments identified a number of job startup scenarios relevant in the workspace context, showing how they may be used in practice. Preliminary application evaluation of workspaces also proved satisfactory and fully realized our expectations for more flexible resource usage.

More work is needed in order to fully assess the usefulness of these ideas. In the short term, we will focus on the privacy and integrity of migrating workspaces. Workspace distribution (delivery to host) can require transferring images of a few gigabytes; this task can be handled by using the presence of an image on a node as a matching criterion as proposed in [15] or by transferring only partial images as in [28]. In addition to performance impact on individual applications, we are considering scalability issues that can be addressed by either using a lighter-weight workspace implementation such as [29] or mapping groups of jobs to one workspace.

## Acknowledgments

This work was supported 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 W-31-109-ENG-38.

## References

1. Czajkowski, K., D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe, *The WS-Resource Framework*. 2004: www.globus.org/wsrf.
2. Andrieux, A., K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, *Web Services Agreement Specification (WS-Agreement) Draft 20*. 2004: https://forge.gridforum.org/projects/graap-wg/.

3. Figueiredo, R., P. Dinda, and J. Fortes. *A Case for Grid Computing on Virtual Machines*. in *23rd International Conference on Distributed Computing Systems*. 2003.
4. Keahey, K., K. Doering, and I. Foster. *From Sandbox to Playground: Dynamic Virtual Environments in the Grid*. in *5th International Workshop in Grid Computing*. 2004.
5. 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)*.
6. *VMware: http://www.vmware.com.*
7. Czajkowski, K., I. Foster, V. Sander, C. Kesselman, and S. Tuecke. *SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems*. in *8th Workshop on Job Scheduling Strategies for Parallel Processing*. 2002. Edinburgh, Scotland.
8. Raman, R., M. Livny, and M. Solomon, *Matchmaking: An Extensible Framework for Distributed Resource Management.* Cluster Computing: The Journal of Networks, Software Tools and Applications, 1999. **2**: p. 129-138.
9. *Workspace Management Service*: http://www.mcs.anl.gov/workspace/.
10. Keahey, K., M. Ripeanu, and K. Doering. *Dynamic Creation and Management of Runtime Environments in the Grid.* in *Workshop on Designing and Building Web Services*. 2003. Chicago, IL.
11. McNab, A., *Grid-Based Access Control for Unix Environments, Filesystems and Web Sites.* Proceeings of the CHEP 2003 conference, 2003.
12. Youssef, S., *Pacman: A Package Manager*. 2004: http://physics.bu.edu/~youssef/pacman/.
13. Goldberg, R., *Survey of Virtual Machine Research.* IEEE Computer, 1974. **7**(6): p. 34-45.
14. Czajkowski, K., I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, *A Resource Management Architecture for Metacomputing Systems*, in *4th Workshop on Job Scheduling Strategies for Parallel Processing*. 1998, Springer-Verlag. p. 62-82.
15. Krsul, I., A. Ganguly, J. Zhang, J. Fortes, and R. Figueiredo. *VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing*. in *SC04*. 2004. Pittsburgh, PA.
16. Allcock, W., *GridFTP: Protocol Extensions to FTP for the Grid.* 2003, Global Grid Forum.
17. Butler, R., D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch, *Design and Deployment of a National-Scale Authentication Infrastructure.* IEEE Computer, 2000. **33**(12): p. 60-66.
18. EU DataGrid, *VOMS Architecture v1.1*. 2003.
19. Fraser, K., S. Hand, R. Neugebar, I. Pratt, A. Warfield, and M. Williamson. *Safe Hardware Access with the Xen Virtual Machine Monitor*. in *OASIS ASPLOS 2004 workshop*. 2004.
20. Keahey, K. and K. Doering, *From Sandbox to Playground: Dynamic Virtual Environments in the Grid.* ANL/MCS-P1141-0304, 2003.
21. Rice, P., I. Longde, and A. Bleasby, *EMBOSS: The European Molecular Biology Open Software Suite Trends in Genetics.* 16, 2000. **6**: p. 276-277.
22. Adabala, S., 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 Generation Computer Systems, 2004.
23. Sundararaj, A. and P. Dinda. *Towards Virtual Networks for Virtual Machine Grid Computing*. in *3rd USENIX Conference on Virtual Machine Technology*. 2004.
24. Jiang, X. and D. Xu, *VIOLIN: Virtual Internetworking on OverLay INfrastructure.* Department of Computer Sciences Technical Report CSD TR 03-027, Purdue University, 2003.
25. Sapuntzakis, C., D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M.S. Lam, and M. Rosenblum. *Virtual Appliances for Deploying and Maintaining Software*. in *Proceedings of the 17th Large Installation Systems Administration Conference (LISA '03)*. 2003.
26. Chase, J., L. Grit, D. Irwin, J. Moore, and S. Sprenkle, *Dynamic Virtual Clusters in a Grid Site Manager.* accepted to the 12th International Symposium on High Performance Distributed Computing (HPDC-12), 2003.

27. Reed, D., I. Pratt, P. Menage, S. Early, and N. Stratford. *Xenoservers: Accountable Execution of Untrusted Programs*. in *7th Workshop on Hot Topics in Operating Systems*. 1999. Rio Rico, AZ: IEEE Computer Society Press.
28. Sapuntzakis, C., R. Chandra, B. Pfaff, J. Chow, M.S. Lam, and M. Rosenblum. *Optimizing the Migration of Virtual Computers*. in *5th Symposium on Operating Systems Design and Implementation*. 2002.
29. Whitaker, A., M. Shaw, and S.D. Gribble. *Denali: Lightweight Virtual Machines for Distributed and Networked Applications*. in *In Proceedings of the USENIX Annual Technical Conference*. 2002. Monterey, CA.