From Sandbox to Playground: Dynamic Virtual Environments in the Grid

Katarzyna Keahey
Argonne National Lab
Argonne, IL
keahey@mcs.anl.gov

Karl Doering
University of California
Riverside, CA
kdoering@cs.ucr.edu

Ian Foster
Argonne National Lab
Argonne, IL
foster@mcs.anl.gov

Abstract

Much experience has been gained with the protocols and mechanisms needed for discovery and allocation of remote computational resources. However, the preparation of a remote computer for use by a distributed application also requires the creation of an appropriate execution environment, which remains an ad hoc and often clumsy process. We propose here a codification of the interactions required to negotiate the creation of new execution environments. In brief, we model dynamic virtual environments (DVEs) as first-class entities in a distributed environment, with Grid service interfaces defined to negotiate creation, monitor properties, and manage lifetime. We also show how such DVEs can be implemented in a variety of technologies—sandboxes, virtual machines, or simply Unix accounts—and evaluate costs associated with these different approaches. DVEs provide a basis for both customization of a remote computer to meet user needs and also enforcement of resource usage and security policies. They can also simplify the administration of virtual organizations (VOs), by allowing new environments to be created automatically, subject to local and VO policy. Thus, DVEs have the potential to relieve much of the current administrative burden involved in providing and using Grid resources.

1 Introduction

The dynamic acquisition and use of remote computers requires policy-driven resource management mechanisms that can establish new computational environments without human intervention [1]. Grid technologies such as the GRAM remote access protocol [2], single-sign-on [3], and agreement negotiation [4] are significant steps towards this goal. However, the problem of establishing and managing execution environments on remote computers remains. The common approach of using static user accounts has high administrative costs and creates environments that do not reflect dynamically changing policies, allow for customized execution environments, or provide QoS enforcement capabilities. Experiments show that virtual machine technology [5-8] can be used to address some of these issues, but no standardized mechanisms have been defined for interacting with such virtual machines.

We believe that the solution to these problems is to introduce abstractions, protocols, and tools that allow remote execution environments to be created and managed as first-class entities. Thus, users will be able to negotiate the creation of new execution environments, administrators will be able to specify the policies that govern their use, and various entities can be authorized for monitoring and management. We expect that in implementing such ideas, we can exploit recent advances in virtual machine and sandbox technologies.

These observations motivate the work presented in this article, which comprises three distinct but interrelated thrusts.

First, we show how dynamic virtual environments (DVEs) can be modeled as Grid services, thus allowing a client to create, configure, and manage remote execution environments using common protocols.

Second, we show how such DVEs can be implemented via a variety of technologies, including dynamic accounts and virtual machines, to obtain access to a range of virtualization and resource management functions. We also examine how DVEs can be implemented within the context of a particular Grid middleware framework, Globus Toolkit 3 (GT3).

Third, we present an experimental evaluation of various DVE implementation technologies. Our results allow us to evaluate the impact of technology choices both in quantitative terms (e.g., computational costs and resource usage) and also respect to other qualitative concerns that arise in Grid contexts.

The dynamic creation of remote environments as user accounts has been previously investigated [9-13], as has the use of virtual machines to model virtual resources [8, 14, 15]. Our work is distinguished by its focus on creating, configuring and managing execution environments as first class entities, that can be implemented via different technologies as dictated by the needs of sites and organizations.

2 Dynamic Virtual Environments

We speak first to the DVE abstraction, its representation in terms of Grid service interfaces, and our prototype implementation within GT3.

Our goal in introducing the DVE abstraction is to codify the interactions required for a client to create, monitor, manage, and ultimately destroy a remote execution environment. Our approach is to model individual DVEs as stateful Web services [16] (in OGSI [17], our focus here) or, as we shall consider in future work, as WS-Resources [18]. We adopt OGSI/WSRF because DVE management operations map conveniently to OGSI/WSRF mechanisms. In particular, OGSI/WSRF lifetime management mechanisms can be used to manage the creation and destruction of DVEs, and OGSI/WSRF state representation and inspection mechanisms can be used to provide access to descriptions of DVE properties such as quality of protection, resource limits, and configuration.

2.1 Creating Dynamic Virtual Environments

DVEs are represented as Grid services and created by DVE factories. As shown in Figure 1, a factory first authorizes the request to create a DVE with the requested properties. An authorization failure results in an exception. On success, the factory performs the following actions: (1) creates a DVE Grid service, (2) initializes its implementation (this could for example involve creating a Unix account or a new J2EE container) and sets its properties (such as its termination time), and (3) records access and other usage policy for the newly created DVE. As a result of the creation process a Grid service handle (GSH) representing the newly created DVE is returned to the client.

DVE creation, configuration, and deployment should in principle be separate. However, in our current prototype DVEs are configured and deployed at creation time. The creation process is securely logged to allow for audit.

DVE termination is managed via the use of OGSI lifetime management mechanisms, which allow the user to request both explicit destruction and implicit (soft-state, or lifetime based) termination. Termination involves cleaning up the state associated with this DVE: policies may be revoked and information relevant to DVE erased. Termination might involve deleting (or returning to a pool) a dynamically created account or virtual machine.

2.2 Dynamic Virtual Environment Services

The DVEService is a Grid interface to a transient, dynamically created execution environment. DVEService shares the properties of any other Grid service: it is identified by a handle, subject to soft-state lifetime management, and exposes its properties (such as the disk space or memory associated with the environment, and/or installed software) through Service Data Elements (SDEs). The interface allows the client to manage the DVE, by for example extending its original termination time, requesting more disk space, or installing software. These requests are authorized in the context of credentials that may be dynamically granted and adjusted [19].

2.3 DVEs and Grid Resource Management

The process of job submission against a DVE is illustrated in Figure 1:

1. The client sends a request for DVE creation to the factory. The request may include the properties and lifetime of the DVE, as well as the client’s credentials.
2. The factory authorizes the request. If the client is not authorized to create the environment as requested, an exception is thrown. Otherwise, a DVE service is instantiated.
3. At instantiation, the DVE service creates an execution environment in an implementation-dependent way. New policy is recorded allowing or restricting access and management of the newly created environment.
4. A handle to the DVE service is returned to the client.

5. At any time during the DVE lifetime, authorized clients may inspect or manage its properties.
6. At any time during the DVE lifetime, authorized clients may perform operations on the DVE (for example, request execution of programs) as authorized by the associated access policy.
7. When the DVE service is destroyed, all associated state is deleted.

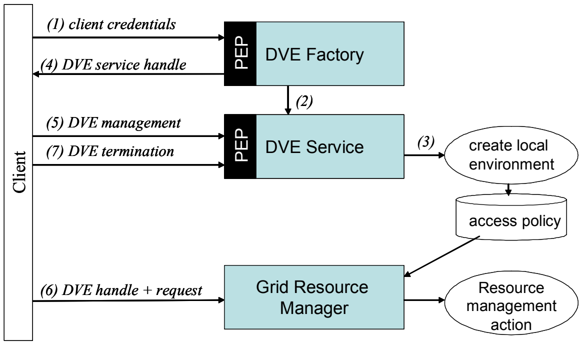

Figure 1: Interactions of DVEs with Grid resource management: the policy evaluation points (PEP) authorize client's requests.

2.4 Implementation of DVEs within GT3

Creating an execution environment on a remote host is a sensitive operation requiring special privileges; it is therefore critical that it should be carried out securely. In our implementation the execution environment is created by the DVE service running under a reserved Unix account (called *globus*). The creation request (as well as management requests on the DVE service) is authorized by an application-specific authorization based on the requestor's Grid credential and a callout to an access control list. The DVE service then uses a setuid program, executable only by *globus*, that validates its arguments and refuses to undertake any actions contrary to a predefined policy (for example, only accounts within a prespecified UID range may be created and destroyed), thus limiting risk to other accounts on the system should the *globus* account be compromised.

The access policy to the execution environment is recorded in the GT3 gridmapfile. Although more complex policies are envisioned, in the current implementation the policy simply gives the right to use the environment to the Grid entity that created it. For the purposes of audit, the creation process is securely logged using the GT3 logging mechanism.

DVE termination implies that the entire DVE state is cleaned in an implementation-specific way. In addition, any state associated with the Grid infrastructure has to be cleaned. We remove access policies from the gridmapfile, and clear entries from the GT3 port mapping file which assists in hosting environment restoration in case it was not cleanly shut down. This eliminates attempts to recreate the environment for nonexistent (or worse yet, different) user.

In order to integrate DVEs with the GT3's Grid Resource Management System (GRAM) [2], we extended the GRAM protocol to include a handle to a specific DVE with the description of remote actions to be performed. For backward compatibility, if the handle is not presented, then the first relevant mapping in the gridmapfile is used. We also changed its implementation to integrate DVE access policies into its authorization mechanism.

Introducing DVEs splits the process of job submission into two phases: creation and deployment of a DVE and job submission against that environment. While creation of a DVE simply replaces the process of obtaining an user account, it can still be seen as complex by users who are interested in executing things quickly on new resources. For this reason, we have also provided a simplified job submission facility in which the user delegates its credentials to GRAM which then automatically obtains an execution environment for the user, executes the request, and terminates the environment. Since the DVE is destroyed, this mechanism does not allow the user to preserve state between different executions but provides a significant simplification.

3 DVE Implementations

We expect that the DVE implementations will be chosen based on the security, performance, cost and flexibility requirements of different sites. We have investigated a range of different options and decided to focus on a group of implementations meeting the following criteria:

- *Generality*: in order to accommodate the largest possible set of codes, the implementation should be generic rather than focused on a specific technology or language (such as the Java Virtual Machine (JVM) [20] for example).
- *Non-invasive*: while techniques such as software fault isolation [21, 22], and proof-carrying code [22] are viable options for DVE implementation they require the system to modifying binary or source of application code which may not be acceptable to some Grid users.

- *Protection*: the DVE implementation should provide suitable levels of protection between users (not allowing users read each other's files for example) as well as between the user and resource (not allowing a user to gain superuser privileges on a resource)
- *Enforcement*: an implementation should offer a range of enforcement options, i.e., allow enforcing disk quotas or CPU share.
- *State*: the ability to preserve state associated with a specific environment, ranging from environment settings to execution state, is an important option of execution environments especially if we consider migrating them between resources.

Given these criteria, our exploration of technologies focused on three kinds of technologies: Unix accounts, sandboxes and virtual machines. Sandboxes provide secure environments restricting executing code to a certain protection environment. Kernel-level sandboxes [23, 24] are efficient, but are possible only where the kernel is available for modification and require sites to run a custom installation of the operating system. User-level sandboxes [25, 26] rely on finding a way to prevent the user from bypassing the interception mechanism which is typically expensive (on the order of 40% for ptrace-based systems [27]). For our evaluation, we selected the VServer technology [7] using kernel-level modifications (also used by PlanetLab [15]). Unlike a sandbox which strives only to confine a user's activity on a given machine to a limited subset of a resource, a virtual machine (VM) provides an abstraction of the physical system itself so that multiple operating system can coexist on the actual machine sharing its resources. In our original experiments, we started out with User Mode Linux (UML) [28] but encountered difficulties running certain Java software packages. VMware [5] proved a more reliable choice.

The sections below contain a description of the implementation of the DVEs for each of the three local enforcement vehicle used: Unix accounts, VServer, and VMware. The implementation aspects include environment creation/deployment, destruction, and the management aspects specific to each technology. Another common feature of interest is how a Grid coordination entity on a given machine (such as a local GRAM installation needed to start jobs) can interact with those implementations (for example: start a user hosting environment). Although we currently deploy DVEs on creation, we also investigated capability of a technology for preserving state to accommodate cases where a DVE could be redeployed in a different setting.

3.1 Unix Accounts as DVE Implementation

Unix accounts [9-13] can be created dynamically using standard systems tools, or allocated from a pool of pre-generated accounts. The former approach has the advantage that accounts can be flexibly created based on need, but requires a secure process for account creation. In addition, depending on the mechanism used, the implementation needs to be careful to respect the assumptions of a local account management system. Pre-generating accounts is limited in the number of users this method can service. Using either method, we want to avoid mapping two different Grid identities to the same local account in order to avoid problems of audit.

In our current implementation, we create accounts on the fly by modifying the system password file. We could also update a NIS/YP password database to create accounts valid across a cluster. To destroy an account, all processes running under the account are killed, all files associated with the account are erased, and then the corresponding entry is removed from the password file. In order to avoid the need for a system sweep every time an account is destroyed, accounts are created with limited write privileges.

The enforcement capabilities offered by standard Unix tools are limited. Disk space usage can be enforced dynamically by using the `quota` command. In addition, `chroot` restricts a user to a subtree of the host filesystem which can be useful if we want to restrict the account to its home directory. The `setrlimit` system call is available for setting more fine-grain limits on maximum CPU time, file size, memory usage and number of open files and processes, but few operating systems fully enforce the limits.

The static user state for a Unix account is relatively easy to manage: files belonging to a certain account (including symbolic links) can be stored in a designated place. The management of execution state would require further support from the local system in the form of checkpointing procedures.

3.2 VServer as DVE Implementation

VServer [7] provides Linux kernel-based virtual servers via the addition of security contexts to the kernel, a small number of new system calls, and management utilities. Inside a VServer security context, processes can only see other processes in the same context, superuser capabilities are restricted, filesystem access may be confined to a subtree of the server's filesystem, and networking and interprocess communication can also be restricted.

Creating a new VServer DVE involves creating a root filesystem for the VServer. To do that, a copy of a precreated minimal filesystem (containing GT3 and supporting software and libraries) is made. To reduce disk space usage (as well as copy time), hard links are used rather than a true copy, and file attributes are set such that a user (including "root") in one VServer cannot modify files shared by other VServers. However, with the immutable attribute set, not only can a file not be modified, but it cannot be removed and replaced with a modifiable copy, either. To remedy this, the VServer kernel patch introduces a new attribute bit, immutable-linkage-invert, which when set allows immutable files to be unlinked, so that VServer users may remove and replace files within their own filesystem tree without affecting other VServers. Once the VServer root filesystem is ready, it can be activated by establishing a corresponding new security context, and launching the user hosting environment inside that context. To destroy a VServer environment, all processes running in the associated security context are terminated, and the corresponding filesystem tree is deleted.

Since all processes associated with a VServer environment share a unique security context ID, the kernel scheduler could potentially be modified to enforce per-environment CPU utilization limits. The current implementation simply prevents one VServer from starving another, but could be extended. Likewise, the possibility exists for adding per-environment memory usage enforcement. Per-environment disk usage is enforced via per-context quotas, an extension of standard user/group quotas. Optionally, network usage is restricted by binding environments to separate IP addresses and then using the Linux kernel firewalling and traffic shaping capabilities exactly as for the VMware DVE implementation.

VServer does not provide capabilities for state management beyond that provided by a Unix account.

3.3 VMware as DVE Implementation

VMware [5] is a commercial virtual machine implementation available in server and workstation versions. For our evaluation we used the workstation version. Physically, the VMware virtual machine consists of a directory on the host containing a set of files, including VM configuration information and virtual disk image(s).

In order to avoid repeating installation information for each DVE creation, we precreated a "master" VMware virtual disk, with a minimal Linux and GT3 installation, which is then used by the individual sessions. The VM-based DVE is activated by a launching script similar to the one used in the Unix account implementation. Inside the VM, at the end of the boot sequence, the user hosting environment is then created. Its creation is complicated by the fact that command-line arguments required by the hosting environment for initialization can no longer be passed to it directly from the launching script, because the launching script and hosting environment are running on two separate machines. For reasons of simplicity, we circumvented this problem by passing the information covertly (by encoding information in the MAC address of the virtual machine, which is modifiable), although implementing a DHCP-like discovery service would provide a more elegant solution.

The deactivation process is complicated by the fact that VMware Workstation provides no published interface for shutting down a particular VM from the host. Our workaround was to run a shutdown service inside the VM, which the host could then contact to shut down the VM. This works as long as the user owning the VM does not have root access inside the VM (otherwise he could kill the shutdown service). Once the VM is no longer running, a DVE may be destroyed by deleting the directory on the host containing the files corresponding to the VM.

VMware provides only static enforcement of memory and disk usage. VMware workstation also does not support any capability for CPU management. (This capability is supported by VMware ESX Server.) If multiple virtual machines are used, VMware can be configured to use NAT/bridged network Since each VM has its own virtual MAC and IP address, the host can do full firewalling and traffic shaping per-VM (for the Linux capabilities, see [29]). In this setting difficulties arise due to the fact that the VM's private IP address and/or port number may be exposed at a higher level in the network protocol stack (for example as part of the Grid Service Handle (GSH)), which standard NAT cannot handle. We managed to deal with them to some extent through careful configuration of the toolkit.

Finally, while VMware does implement a promising solution for user state preservation and restoration, we were unfortunately not able to take advantage of it because the product does not expose a protocol for accessing it but exposes it only through a Graphical User Interface (GUI).

4 Comparison

In addition to qualitative comparison of different implementations, we conducted a quantitative analysis estimating the impact of different implementations on the user programs run within those environments as well as their efficiency of resource usage. We conducted our evaluation on a Pentium 4 3.0 GHz machine with 1 GB RAM, running Red Hat Linux 9.0 (kernel 2.4.22, gcc 3.2.2) and Globus Toolkit 3.0.2. We implemented DSVs using Linux accounts, VServer 0.26, and VMware Workstation 4.0.5.

We first estimated the impact on the performance using as benchmark a scientific compute-intensive application called EFIT [30] representative of the needs of one of the experimental communities using Grids. The application is mostly CPU-bound with minimal access to disk.

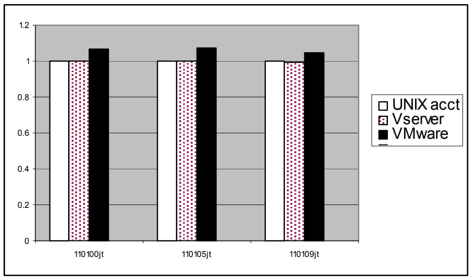

Figure 2: Comparison of different EFIT runs executing directly on Unix, under VServer and VMware. The times on Y axis are normalized to the Unix run.

The results summarized in Figure 2 show execution times directly over the operating system, under VServer, and under VMware. Each grouping represents runs for different dataset and application configurations. For each run, the data shows the mean of five runs per selected data set, normalized to the execution time directly on the operating system. The runs took approximately 100 sec each, with standard deviations between 0.2 sec and 1.2 sec. VMware runs took 6% longer; there was no statistically significant difference between direct operating system runs and VServer runs. The time shown includes the application execution time only, as measured via the gettimeofday() system call, and does not include DVE creation times. The measurements confirm the result presented in [8]; the performance impact of the selected enforcement technologies on compute bound applications is relatively small.

We next compared create/destroy times for the different technologies as shown in Table 1. Measurements do not include time taken to invoke the factory service, which is independent of the implementation type, but simply represents the time elapsed for the implementation-specific callout for environment creation (as measured within theservice).

Table 1: DVE create/destroy times

	Linux	VServer	VMware
Create	100 ms	360 ms	14-52 sec
Destroy	70 ms	200 ms	3-38 sec

Creating both VServer and VMware DVEs involves creating a new file system root. Our file system size was just under 300 MB, including: (1) minimal Red Hat 9.0 (with stripped-down /bin, /dev, /etc, /lib, /proc, /sbin, /var): 14,636 KB, (2) Perl 5.6.1 (used by GT3 job execution components): 29,728 KB, (3) Java 2 Runtime Environment 1.4.2: 60,320 KB, and (4) Globus Toolkit 3.0.2: 188,408 KB. Because VServer uses a copy-on-write technique, environment creation time as compared to that of VMware is drastically reduced. Another interesting measure is the overhead used by these technologies, in other words, a measure of how efficiently they use the resource. The results are summarized in Table 2.

For VServer, there is in fact a very small amount of memory and CPU overhead resulting from the VServer kernel modifications: the kernel must track which security context a process is associated with, and incurs slight overhead in checking the security context inside relevant system calls. However, this overhead was sufficiently small as to not show up in our measurements, and is therefore listed as negligible. While both technologies induce some overhead of resource usage, for VMware this overhead is significantly larger.

Table 3 summarizes the qualitative differences among the different technologies. Both VServer and VMware offer substantial improvement over plain accounts in terms of protection and sharing. VServer allows the creation of separate security contexts which restrict user privileges but allow for sharing of files. However, all contexts still share the same kernel. VMware allows each execution environment to run its own kernel. This requires repeating all required software installations for each VM running on the machine: an inconvenience compounded by potential licensing issues. VServer offers dynamic enforcement capabilities (or potential of such

Table 2: Resource overheads (over Linux) of DVE implementations

	VServer	VMware
Disk overhead	Small: approximately 0.5%	Large: 150% - 200%
Memory overhead	Negligible	Large: 24MB + 1 MB per 32 MB memory allocated per VM
CPU overhead	Negligible	Depends on application characteristics (5% and up)
Network overhead	Only when restricting access	Yes: depends on network configuration

Table 3: Enforcement capabilities of selected DVE implementations

	Unix account	VServer	VMware
CPU usage (seconds)	Via setrlimit()	Not at present, but could be added	Not enforced
CPU usage (percent)	Not enforced	Limited: no VServer can starve another	Not in VMware Workstation, but enforced in VMware Server
Disk space usage	Dynamically (per-user quotas)	Dynamically (per-context quotas)	Statically (virtual disks)
Memory usage	No	Not at present, but could be added	Statically
Network usage	No	Dynamically (binding contexts to specific IP addresses)	Dynamically (via VM configuration and host firewall)

capabilities), VMware has only limited static enforcement capabilities. This however is not the case for other virtual machine technologies [6]. Although VServer presents a more lightweight solution in terms of performance, its impact, at least in our experience, is not significant and can be expected to decrease as the technology improves. To balance this, virtual machines offer the potential for better user state management, which could have significant benefits for implementing migration in the Grid environment.

5 Conclusions

We strongly believe that in order to become successful Grids will need the ability to create and manage remote execution environments dynamically and effortlessly. Rather than imposing one implementation, these environments should be able to rely on a mix of technologies as acceptable to site administrators, users and suitable for specific problems. Different implementations will provide different functionality ranging from a simple execution environment to a high degree of customization (including running the required operating system kernel on any resource possible through the agency of true virtual machines). Different implementations entail different trade-offs in terms of efficiency, cost, configurability, quality of protection and other characteristics as discussed above.

Our exploration of three such implementations shows that all are roughly acceptable from the point of view of efficiency for a Grid application without strong I/O demands. All have shortcomings in the area of QoS enforcement, but those could potentially be fixed by similar technologies or more advanced versions of the same software. The issue of sharing between the environments presents an interesting trade-off: on one hand virtual machines allow users to customize their environments once and then port them across different machines, on the other this practice will lead to bulky installations and potential duplication of much of the software on one real resource. In those cases, software allowing sharing between environments, such as VServer, can lead to more lightweight solutions, but dependent on a shared infrastructure maintained on a resource by, for example, the VO.

Finally, it is clear that some of the current execution environment examples were not designed with Grids in mind (the lack of exposed protocols in VMware workstation is an example). While more research is necessary in order to fully determine the requirements for an ideal sandbox implementation to put in the Grid playground, the Grid technologies themselves will also have to change. The widespread use of remotely created virtual environments, if successful, will shift trust from the account screening process typically applied by the resource owner to screening process implemented by a virtual organization which will thereby acquire more importance as well as responsibility.

6 Acknowledgement

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, SciDAC Program, U.S. Department of Energy, under Contract W-31-109-ENG-38.

7 References

1. Foster, I., *What is the Grid? A Three Point Checklist*. 2002: http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf.
2. 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.
3. 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.
4. Czajkowski, K., A. Dan, J. Rofrano, S. Tuecke, and M. Xu, *Agreement-based Grid Service Management (OGSI-Agreement) Version 0.* https://forge.gridforum.org/projects/graap-wg/document/Draft_OGSI-Agreement_Specification/en/1/Draft_OGSI-Agreement_Specification.doc, 2003.
5. *VMware*: http://www.vmware.com/.
6. 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)*.
7. Solucorp, *vserver*: http://www.solucorp.qc.ca/miscprj/s_context.hc.
8. Figueiredo, R., P. Dinda, and J. Fortes. *A Case for Grid Computing on Virtual Machines*. in *The 23rd International Conference on Distributed Computing Systems (ICDCS)*. 2003.
9. Hacker, T. and B. Athey, *A Methodology for Account Management in Grid Computing Environments.* Proceedings of the 2nd International Workshop on Grid Computing, 2001.
10. Kapadia, N.H., R.J. Figueiredo, and J. Fortes. *Enhancing the Scalability and Usability of Computational Grids via Logical User Accounts and Virtual File Systems*. in *10th Heterogeneous Computing Workshop*. 2001. San Francisco, California.
11. Talwar, V., S. Basu, and R. Kumar. *An Environment for Enabling Interactive Grids*. in *The Twelfth IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*. 2003. Seattle, Washington.
12. McNab, A., *Grid-Based Access Control for Unix Environments, Filesystems and Web Sites.* Proceeings of the CHEP 2003 conference, 2003.
13. 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 (to appear)*. 2003. Chicago, IL.
14. 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.
15. Bavier, A., M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. *Operating System Support for Planetary-Scale Services*. in *Proceedings of the First Symposium on Network Systems Design and Implementation (NSDI)*. 2004.
16. Foster, I., C. Kesselman, J. Nick, and S. Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. 2002: Open Grid Service Infrastructure WG, Global Grid Forum,.
17. S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, and C. Kesselman, *Grid Service Specification*.
18. Foster, I., J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, T.

Storey, W. Vambenepe, and S. Weerawarana, *Modeling Stateful Resources with Web Services.* www.globus.org/wsrf, 2004.

19. Foster, I., C. Kesselman, and S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations.* International Journal of High Performance Computing Applications, 2001. **15**(3): p. 200-222.

20. Lindholm, T. and F. Yellin, *The Java(TM) Virtual Machine Specification (2nd Edition)*. 1999: Addison-Wesley Pub Co; 2nd edition.

21. Wahbe, R., S. Lucco, T. Anderson, and S. Graham, *Efficient software-based fault isolation*, in *Proc. 14th Symposium on Operating System Principles*. 1993.

22. Necula, G.C. and P. Lee. *Safe Kernel Extensions without Run-Time Checking*. in *2nd Symposium on Operating Systems Design and Implementation*. 1996. Seattle, WA.

23. Cowan, C. and D. Wagner, *Linux Security Module.* http://lsm.immunix.org.

24. Loscocco, P. and S. Smaller. *Integrating Flexible Support for Security Policies into the Linux Operating System*. in *FREENIX Track of the 2001 USENIXS Annual Technical Conference*. 2001.

25. Goldberg, I., D. Wagner, R. Thomas, and E. Brewer, *A Secure Environment for Untrusted Helper Applications --- Confining the Wily Hacker*, in *Proc. 1996 USENIX Security Symposium*. 1996.

26. Alexandrov, A.D., P. Kmiec, and K. Schauser. *Consh: A Confined Execution Environment for Internet Computations*. in *USENIX Annual Technical Conference*. 1999.

27. Bosilca, G., F. Cappello, A. Djilali, G. Fedak, T. Herault, and F. Magniette, *Performance Evaluation of Sandboxing Techniques for Peer-to-Peer Computing*. 2002, LRI-CNRS and Paris-Sud University.

28. Dike, J. *A User Mode Port of the Linux Kernel*. in *USENIX Annual Linux Showcase and Conference*. 2000. Atlanta, GA.

29. *Linux Advanced Routing and Traffic Control*: http://lartc.org.

30. Lao, L.L., H. St. John, R.D. Stambaugh, A.G. Kellman, and W. Pfeiffer, *Reconstruction of Current Profile Parameters and Plasma Shapes in Tokamaks.* Nucl. Fusion, 1985. **25**: p. 1611.