Deploying Virtual Machines as Sandboxes for the Grid

Sriya Santhanam*, Pradheep Elango, Andrea Arpaci-Dusseau, Miron Livny
University of Wisconsin-Madison
{sriya, pradheep, dusseau, miron}@cs.wisc.edu

Abstract

The ability to securely run arbitrary untrusted code on a wide variety of execution platforms is a challenging problem in the Grid community. One way to achieve this is to run the code inside a contained, isolated environment, namely a "sandbox". Virtual machines provide a natural solution to the security and resource management issues that arise in sandboxing. We explore different designs for the VM-enabled sandbox and evaluate them with respect to various factors like structure, security guarantees, user convenience, feasibility and overheads in one such grid environment. Our experiments indicate that the use of on-demand VMs imposes a constant startup overhead, with I/O-intensive applications incurring additional overheads depending on the design of the sandbox.

1 Introduction

Security vulnerabilities in grids have been the focus of a lot of attention in recent years. Butt et al. [4] discuss the inadequacy of traditional safety checks, while Miller et al. [13] emphasize the susceptibility of current grid systems by demonstrating a security exploit on a grid using dynamic code instrumentation techniques. Though such specific problems can be tackled, they indicate the dangers of grid applications leaving contaminating residual state. Further, common attacks such as root exploits and stack smashing can always bring down a machine and are a major worry for widely-used grid environments like Condor [8].

To address these concerns, a grid computing system must enable the execution of untrusted, unverified applications within a secure, isolated environment, namely a "sandbox". Essentially, the impact of the running application must be restricted to the sandbox, thereby protecting the underlying host machine. Depending on the application's requirements, access to resources on the host machine can be selectively allowed from within the sandbox. Moreover, while security attacks and faults can occur within the sandbox, its framework must guarantee that these vulnerabilities do not affect the underlying execute machine.

In this paper, we explore the use of Virtual Machine (VM) technology to implement sandboxes. VMs present the image of a dedicated raw machine to the grid application. An application running on a VM is decoupled from the system software of the underlying host machine. This ensures that an untrusted user or application can only compromise the VM and not the underlying physical resource. VMs also enable fine-grained resource allocation for grid jobs. It is hence feasible to restrict the memory, network, disk size, and even the CPU cycles allocated to a given VM. Furthermore, the use of VMs allows the target execution environment for a grid application to be completely customized, thereby enabling support for jobs with special requirements like root access or legacy dependencies. VMs also enable process migration without requiring any modification or relinking of the grid application.

As a result, considerable attention has been focused on the idea of integrating VMs into the grid architecture [5, 9, 11, 19]. However, the notion of the "sandbox" that these VMs are used to create is not uniformly defined, particularly with respect to its structure. A sandbox is defined both by what it contains and where its boundaries are. VMs offer the flexibility to implement different sandbox designs and in this paper, we attempt to explore and evaluate various implementations of the VM-based sandbox along factors such as structure, security guarantees, feasibility, user convenience and performance overheads.

Here we present four different sandbox designs, ranging from a simple substitution of a grid node with a virtual node all the way up to on-demand VMs that use lazy file-retrieval techniques. The choice of sandbox design, as we emphasize in this effort, is workload dependent and our aim is to give a clear understanding of the pros and cons of each solution. We chose to implement our sandboxes on Condor [12], a large, distributed grid environment that is currently deployed across the globe more than 1700 Condor pools, representing

*Author is currently working at VMware, Inc. This work was done while the author was studying at the University of Wisconsin-Madison.

over 60,000 computers. Our sandboxes are deployed using Xen [3] as the virtual machine monitor, which in contrast to other virtual machine monitors that support full virtualization, has been proven to have near-native Linux performance with very low overheads.

2 Current Approaches

VMs are not the only solutions to the problem of sandboxing. Simpler abstractions such as chroot() have been used for the same purpose. However, such sandboxes are limited as they can isolate only the filesystem parts, and further, simple exploits for breaking chroot() are well-known [7]. Programming language virtual machines such as the Java Virtual Machine support similar goals of sandboxing and portability. However they severely restrict the range of applications that can run on them. Jails [10] and Pods [14] are other abstractions that could provide sandboxing. The jail mechanism is common in the FreeBSD world, and is a stronger variant of chroot() sandboxes. Jails, however, do not provide much support for other desirable properties such as migration. Pea-pods [15], an extension of Pods, support migration although this migration is restricted to different versions of the same kernel. While adapting such mechanisms for the grid environment could open up interesting research avenues, we choose to work with VMs owing to the several benefits they provide along with the flexibility they offer for different sandboxing solutions.

The idea of integrating VMs with grids was initially popularized by Figueriredo et al.[9], whose observations subsequently led to the INVIGO project[1]. However, the INVIGO system addresses higher level issues such as service discovery and composition, paying relatively lesser attention to lower-level issues like security and isolation. Further, INVIGO uses a VM monitor that supports full virtualization, while we use a paravirtualized monitor which would incur much lesser overheads. Entropia [5] and SETI@Home[16] address the issue of sandboxing using binary rewriting solutions that only support custom-developed grid applications; thus they restrict the range of applications that can run on their grid.

3 Design and Implementation

3.1 Sandbox 1: Virtual grid nodes

This is the simplest sandbox in terms of design and implementation. A virtual machine joins the grid just like any other physical machine. It serves as a substitute for the underlying physical machine in the grid.

Even this simple design however offers most of the benefits of VM technology, as summarized by Table 1. For instance, a node can be easily configured to control resource allocation for the grid applications, restricting their use of main memory, disk usage, swap space and CPU cycles. Further, since all resources are virtualized on this node, its configuration can be different from that on the physical node; for example, the network card on the VM could be restricted by firewall policies that are entirely different from that on the physical machine. Virtual nodes also provide isolation from the other applications that are running on the physical machine. Isolation can prevent attacks from residual processes of a grid job that could have run previously, in addition to preventing security attacks and faults from spreading over to the physical machine.

In our implementation, we replace the execute machine that Condor sees with a Xen VM. The VM has network access and is typically assumed to be running in the background on the execute machine during normal operation. Condor is installed only on the VM and not on the underlying physical machine. This ensures that jobs run only inside the VM and are never directly executed on the physical machine. The only drawback of this design is that the application is still open to the network, and can hence launch network attacks on any accessible system.

3.2 Sandbox 2: Eager prefetching, whole-file-caching sandboxes

In this design, VMs act as individual job containers with no network access. To guarantee correctness while disallowing network access, the VM must contain all job requirements within the sandbox before the job begins execution. It does this by eagerly fetching all the data files needed by the job prior to execute time. All I/O requests made by the job at run time will hence be satisfied locally within the sandbox.

This design uses VMs as ad-hoc entities that exist only for the duration of the job. While this incurs the overheads of starting and shutting down a virtual machine for each run of a grid job, on the flip side, system resources are consumed only for the job duration. This solution also allows the flexibility of launching a VM with a pre-configured environment that matches the job's requirements, although since job dependencies have to be identified by the user beforehand, there is a downside on user convenience. However, for applications with well-defined I/O behavior, this design provides the tightest guarantees of sandboxing.

3.3 Sandbox 3: Lazy, block-caching sandboxes

In this design, the boundary of the sandbox is extended to include the original machine from which

the job was submitted, i.e. the "submit machine". System calls are trapped and executed remotely at the submit machine; only the results are transferred back to the execute machine. This avoids any library or software compatibility issues as the applications are not tied to the software configuration of the underlying physical system.

This solution will be particularly useful when an application has huge input file dependencies but makes few I/O calls. As before, VMs are launched on-demand. User convenience is enhanced since the user need no longer specify job dependencies. However, since the VM is configured with restricted network access, limited network attacks are still possible. The security guarantees provided by this sandbox are hence not as tight as those provided by Sandbox 2. Sandbox 4 hence attempts to minimize this tradeoff and bring about the best of both worlds.

3.4 Sandbox 4: Lazy prefetching, whole-file caching sandboxes

This design is very similar to our second sandbox design with the main difference being that the decision to prefetch the files required by the job is done dynamically at job run time rather than statically prior to the execution of the job. Hence, this sandbox only transfers entire files when it sees that a file being opened exists on the submit machine. However, in order to transfer data over the network, we need to open up network access to the sandbox, which in turn creates an opportunity for the job to attack/be attacked on the network. Hence, to prevent network attacks in this scenario, the sandbox must suspend the executing job for the time period during which the network is opened.

Our implementation uses Chirp [17], a lightweight system for performing file I/O over a network. Chirp allows file access to be set up with fine-grained control so that user permissions are not violated. Another component of this implementation is Bypass, an application that is used to create interposition agents and split execution systems. This implementation launches a VM that has no network access. All open system calls from the Condor job are interposed. An open call that requests I/O on the network, forces the job to be suspended and a restricted network connection to be opened up. The requested file is then copied to the local machine. Network access is then disabled, and the request proceeds normally. A log maintains new changes to the directory structure, which are merged onto the submit machine after the job completes.

User convenience and flexibility in this scenario are achieved because the user does not have to specify any dependencies for the job. Hence this solution servers as a tradeoff between between designs 2 and 3, offering the tighter security guarantees of sandbox 2 while maintaining the degree of user convenience offered by sandbox 3.

3.5 Suspend/Resume

For all the on-demand VM solutions, it is also possible to preserve process state when the Condor job needs to be vacated from the execute machine. Under normal conditions, Condor jobs are vacated from an execute machine once the machine detects keyboard activity and ceases to be idle. However, since jobs are now running inside a VM, it is possible to suspend and resume the VM on a different machine rather than lose the progress made by the job by restarting it on another machine. This mechanism, already supported by the virtual machine monitor, helps preserve job state across different machines.

To implement this functionality, the wrapper application sets up a signal handler to trap the SIGTERM signal, which is the signal sent by the Condor daemon when it wants to vacate a job. The wrapper then suspends the VM state to a restore file on receipt of this signal, and subsequently exits. Condors file transfer mechanisms are set up to transfer all files written by the job on eviction or exit from the execute machine. Hence, the next time the job is scheduled to run on a (possibly different) machine, the wrapper checks for the presence of the restore file, and if present, resumes the VM rather than booting it up again. The Condor job continues seamlessly from the point of eviction.

4 Evaluation

We characterize the different sandbox designs with respect to factors such as security guarantees, application functionality and user convenience in Table 1. For our experiments, we have chosen to focus on data-centric micro-benchmarks, since applications of this category form a major share of grid workloads [2, 18].

Further, in a virtualized environment like Xen, data-intensive workloads entail a significant CPU overhead [6] and are hence likely to be most affected by our sandboxing techniques. Computation-intensive workloads on the other hand, on account of their minimal interaction with the OS, tend to perform competitively with negligible overhead on the Xen VMM [3]. Data-centric workloads could vary in their characteristics - in the number of files accessed, amount of data transferred as well as the burstiness of I/O rates [18]. We present the results of the experiments that highlights the performance overheads and scalability of

Feature	Condor Vanilla Universe [1]	Condor Standard Universe [1]	Sandbox 1	Sandbox 2	Sandbox 3	Sandbox 4
Security Guarantees						
Protecting execute machine from malicious/faulty job	No	No	Yes	Yes	Yes	Yes
Preventing network attacks by a malicious job	No	No	No	Yes	Limited	Yes
Protecting against lurker processes (one Condor job attacking another)	No	No	No	Yes	Yes	Yes
Application Functionality						
Customizing job execution environment (Libraries, Packages, etc)	Limited	Limited	Limited	Yes	Yes	Yes
Running jobs with network requirements	Yes	Yes	Yes	No	Limited	No
Running jobs which require higher privileges (e.g. root access)	No	No	Yes	Yes	Yes	Yes
Running jobs which require a different OS than the one on the execute machine	No	No	Yes	Yes	Yes	Yes
Reflects all writes made by the job	No	Yes	Only in Standard Universe	No	Yes	Yes
Administrator and End-user Usability						
Controlling resource allocation (CPU, memory, disk) on execute machine	Limited	Limited	Yes	Yes	Yes	Yes
Job Migration	No	Yes	Only in Standard Universe	Yes	Yes	Yes
Requires user to specify job requirements	Yes	No	Only in Vanilla Universe	Yes	No	No

Table 1: *Comparison of the four sandboxes with respect to various metrics*

the different designs below.

4.1 Platform

To implement our sandboxes, we used Xen 2.0 on Linux 2.6.9 kernels which run Condor 6.7.6. Condor [12] is a large, distributed grid environment that is currently deployed across the globe more than 1700 Condor pools, representing over 60,000 computers. All our experiments were performed on Dell workstations: 2.6GHz Xeon processors, each using a 120GB Seagate IDE hard disk (7200-RPM, 8.5 ms avg seek time). While a few of the experiments were conducted using the virtual network provided by Xen, in order to take real network overheads into account, we conducted certain experiments over a Gigabit ethernet. The VMs were configured with 128 MB memory and 2GB (virtual) hard disk capacity.

[1] Condor allows two types of jobs, "standard" and "vanilla". Standard jobs can be checkpointed and migrated from system to system transparently by Condor without restarting. However, for a code to be submitted as a standard job it must be recompiled using various Condor-specific compiler options and libraries.

4.2 Experiment 1: A comparison of the four sandboxes

In this experiment, we ran jobs performing 40 million random I/O calls over a physical network, on each of the sandbox designs. Figure 1 shows that Sandbox 1 performs equivalently with Sandbox 2 when jobs are submitted over the network. The worst performing sandbox is Sandbox 3, on account of all 40 million calls being issued remotely. The additional run time overhead incurred by Sandbox 4 is on account of the network transfer delay for the input file. Queuing delays are the highest for Sandbox 3 on account of its implementation requiring three job submissions. For the remaining experiments, we focus our comparison on jobs running on Condor Standard Universe, Sandbox 3 and Sandbox 4 since it is only in these environments that the job runtime involves network transfer overheads that are comparably affected by the amount of I/O performed by the job. In Condor Vanilla Universe, Sandbox 1 and Sandbox2, all the files required by the job are transferred prior to job execution and all I/O is performed locally; job runtimes in these environments are hence not affected

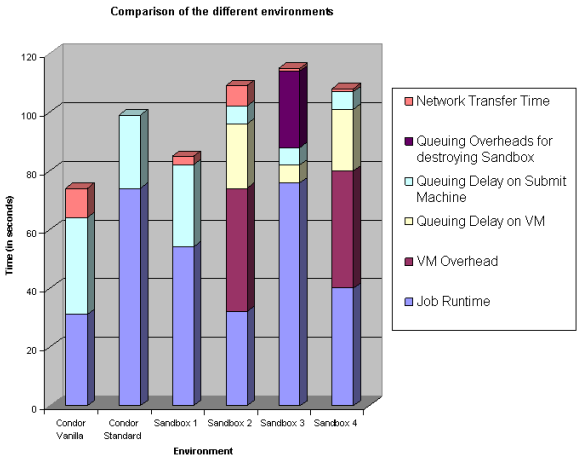

Figure 1: **Experiment 1.** *A comparison of execution times of jobs on the different sandboxes. Condor Vanilla and Condor Standard are the base Condor environments without sandboxing.*

materially by variations in I/O patterns.

4.3 Experiment 2: Varying the input file size

In this experiment, we vary the size of the input file required by a job and demonstrate the effects. Each experiment issues 10,000 reads on an input file the size of which has been varied from 4K to 256M. All reads are random reads, so prefetching and caching effects are factored out. Sandbox 3 and Standard universe jobs transfer all I/O requests over the network. Network overheads are a little higher in Sandbox 3 due to virtualization effects and our measures to guarantee security. Figure 2 shows that both Sandbox 3 and Standard universe jobs follow similar trends. While there is a fixed cost on Sandbox 3 jobs, Sandbox 4 transfers the files during run time. The above graph shows that Sandbox 4 can be a really useful option for I/O intensive jobs. The huge hit in the run time for a 256 MB file is on account of thrashing effects as our VM memory size was limited to 128 MB. The first two sandbox designs are not included in this comparison because the job runtime will not factor in the transfer time for the file.

4.4 Experiment 3: Varying the number of input files

In this experiment, we vary the number of input files required by a job. On every file, we issue 10,000 random read calls. The size of each file was 4 MB. Figure 3 shows that sandbox 3 and the standard universe scale smoothly since they only transfer individual I/O requests instead of entire files.

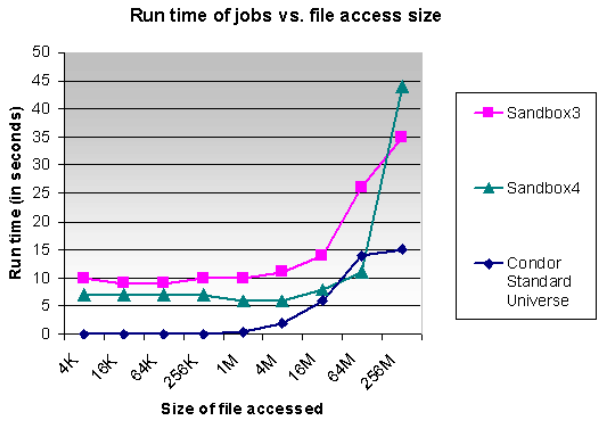

Figure 2: **Experiment 2.** *The effect of input file size on run times.*

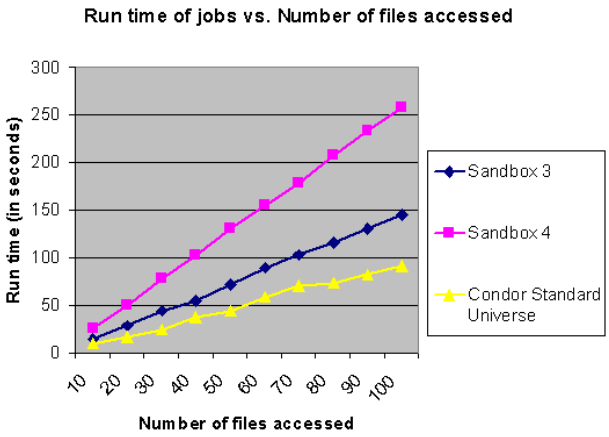

Figure 3: **Experiment 3.** *Run time vs Number of files accessed.*

Sandbox 4 however scales a lot more steeply, since it fetches whole files from the submit machine on each open call. Sandbox 3 hence favors scenarios which involve a lot of input files with relatively fewer I/O operations.

4.5 Overheads of Suspend/Resume

Our implementation of process migration for ad-hoc VM-based sandboxes was built using Xen's suspend/resume capabilities. In our experiments, on average it took 5 seconds for a Xen VM configured with 128 MB of RAM to suspend and about 7 seconds for the same VM to be restored. For this configuration, the size of the suspend file created by Xen and transferred by Condor was 92 MB, which took on average 15 seconds to transfer over the network. VM disk images were cached on the execute machine and hence only the sus-

pend file was transferred across the network. These figures remain static across multiple workloads; a short-lived job running inside the VM will take the same amount of delay to suspend as compared to a long-running application with several pending I/O requests. In general, whenever process migration is possible using mechanisms such as Condor's checkpointing for its Standard Universe jobs, it should be preferred over VM migration since VM migration is a more heavy weight operation. However, this provides a useful alternative for applications that cannot otherwise be checkpointed.

5 Conclusions

Virtual machines prove to be a natural platform for sandboxing in grids, offering a host of benefits like fine-grained resource control and allocation, fault isolation, customized execution environments, and support for process migration among others. The downside of using virtual machines for this purpose is the performance overhead characterized by our experiments. However, given that applications submitted to run in a grid environment like Condor must tolerate delays on the order of several minutes, we believe that these overheads should be acceptable by users who desire the benefits of this approach.

Table 1 highlights and contrasts the features of the different sandbox designs. As indicated, each design offers different levels of security and user convenience. One interesting challenge is the ability to support jobs requiring legitimate network access in a sandbox, while protecting all the hosts on the network. Although we have explored four different styles of sandboxes in this effort, VMs offer very flexible solutions to the problem of sandboxing and hence there are other possible designs that may suit different grid environments. Among our chosen implementations, Sandbox 4 offers the most in terms of flexibility but takes a hit in performance if the workload is I/O intensive. As a limited yet simple solution to sandboxing, Sandbox 1 offers an easy alternative as well.

In the future, we expect to see virtual machines as first class objects integrated into core grid architectures for purposes of sandboxing.

References

[1] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu. From Virtualized Resources to Virtual Computing Grids: The In-VIGO System, Generation Computing Systems, Special Issue, Complex Problem-Solving Environments for Grid Computing, David Walker and Elias Houstis, Editors, June 2005.

[2] G. Allen, T. Goodale, M. Russell, E. Seidel, and J. Shalf. Classifying and Enabling Grid Applications. In *Berman, F., Fox, G.C., Hey, A.J.G. (Eds.); Grid Computing, Making the Global Infrastructure a Reality. Wiley, Chapter 23*, 2003.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[4] A. Butt. Grid-computing Portals and Security Issues. In *Journal of Parallel and Distributed Computing, Academic Press, Inc. Orlando, FL, USA, 2003*, 2003.

[5] B. Calder, A. A. Chien, S. Elbert, and K. Bhatia. Entropia: architecture and performance of an enterprise desktop grid system. In *Journal of Parallel and Distributed Computing, Volume 63(5), pp 597-610*, 2003.

[6] L. Cherkasova and R. Gardner. Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor. In *Proceedings of the USENIX Annual Technical Conference*, April 2005.

[7] How to break out of a chroot() jail. http://www.bpfh.net/simes/computing/chroot-break.html .

[8] Bruce Beckles, Security Concerns with Condor: A brief overview, http://www.nesc.ac.uk/esi/events/438/ security_concerns_overview.pdf.

[9] R. J. Figueriredo, P. A. Dinda, and J. A. Fortes. A Case For Grid Computing on Virtual Machines. In *Proceedings of the International Conference on Distributed Computing Systems(ICSDS)*, May 2003.

[10] P. H. Kamp and R. N. M. Watson. Jails: Confining the omnipotent root. In *The FreeBSD Project, http://phk.freebsd.dk/pubs/sane2000-jail.pdf*, 2000.

[11] K. D. Katarzyna Keahey and I. Foster. From Sandbox to Playground: Dynamic Virtual Environments in the Grid. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID '04)*, November 2004.

[12] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor : A hunter of idle workstations. In 8th International Conference on Distributed Computing Systems, pages 104–111, Washington, D.C., USA, June 1988. IEEE Computer Society Press, 1988.

[13] B. P. Miller, M. Christodorescu, R. Iverson, T. Kosar, A.Mirgorodskii, and F. Popovici. Playing inside the black box: Using dynamic instrumentation to create security holes, parallel process. lett. 11 (2,3) (2001) 267 - 280, 2001.

[14] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The design and implementation of Zap: a system for migrating computing environments. *SIGOPS Oper. Syst. Rev.*, 36(SI):361–376, 2002.

[15] S. Potter, J. Nieh, and D. Subhraveti. Secure Isolation and Migration of Untrusted Legacy Applications. In *Columbia University Technical Report CUCS-005-04*, January 2004.

[16] SETI Institute, Online, http://www.seti-inst.edu/science/setiathome html.

[17] D. Thain. Chirp User's Manual. http://www.cse.nd.edu/ ccl/software/manuals/chirp.html, 2004.

[18] D. Thain, J. Bent, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and M. Livny. Pipeline and Batch Sharing in Grid Workloads. In *Proceedings of High-Performance Distributed Computing (HPDC-12)*, page 152161, June 2003.

[19] M. Zhao, J. Zhang, and R. J. Figueriredo. Distributed File System Support for Virtual Machines in Grid Computing. In *Proceedings of the High Performance Distributed Computing (HPDC)*, July 2004.