A Case For Grid Computing On Virtual Machines

Technical Report TR-ACIS-02-001

August 2002

Advanced Computing and Information Systems Laboratory

Electrical and Computer Engineering - University of Florida

Gainesville, FL, USA 32611-6200

Renato J. Figueiredo
Department of Electrical and Computer Engineering
University of Florida
renato@acis.ufl.edu

Peter A. Dinda
Department of Computer Science
Northwestern University
pdinda@cs.northwestern.edu

José A. B. Fortes
Department of Electrical and Computer Engineering
University of Florida
fortes@acis.ufl.edu

[0] Effort sponsored by the National Science Foundation under Grants EIA-9975275, ANI-0093221, ACI-0112891, EIA-0130869, EIA-0224442, ANI-0301108. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation (NSF).

Contents

List of Tables

List of Figures

Abstract

We advocate a novel approach to grid computing that is based on a combination of "classic" operating system level virtual machines (VMs) and middleware mechanisms to manage VMs in a distributed environment. The abstraction is that of dynamically instantiated and mobile VMs that are a combination of traditional OS processes (the VM monitors) and files (the VM state). We give qualitative arguments that justify our approach in terms of security, isolation, customization, legacy support and resource control, and we show quantitative results that demonstrate the feasibility of our approach from a performance perspective. Finally, we describe the middleware challenges implied by the approach and an architecture for grid computing using virtual machines.

1 Introduction

The fundamental goal of grid computing [17] is to seamlessly multiplex distributed computational resources of *providers* among *users* across wide area networks. In traditional computing environments, resources are multiplexed using the mechanisms found in typical operating systems. For instance, accounts are needed to multiplex access to an entire computer, time-sharing enables the multiplexing of processors, and file systems multiplex storage. These and other traditional multiplexing mechanisms assume that trust and accountability are established by a centralized administration entity. In contrast, multiplexing in a grid environment must span independent administrative domains, and cannot rely on a central authority.

The level of abstraction upon which current grid middleware solutions are implemented is that of an *operating system user*. This approach suffers from the limitations of traditional user account models in crossing administrative domain boundaries [19]. In practice, multiplexing at this level of abstraction makes it difficult to implement the security mechanisms that are necessary to protect the integrity of grid resources from untrusted, legacy codes run on general-purpose operating systems by untrusted users [7]. It also greatly complicates the management of accounts and file systems that are not suited for wide-area environments [14]. Unfortunately, most applications need precisely these services.

We propose to fundamentally change the way grid computing is performed by raising the level of abstraction from that of the operating system user to that of the *operating system virtual machine or VM* [23]. This addresses three fundamental issues: support for legacy applications, security against untrusted code and users, and computation deployment independently of site administration.

Virtual machines present the image of a dedicated raw machine to each user. This abstraction is very powerful for grid computing because users then become strongly decoupled from a) the system software of the underlying resource, and b) other users sharing the resource. In terms of security, VMs ensure that an untrusted user or application can only compromise their own operating system within a virtual machine, not the computational resource (nor other VMs). In terms of administration, virtual machines allow the configuration of an *entire* operating system to be independent from that of the computational resource; it is possible to completely represent a VM "guest" machine by its virtual state (e.g. stored in a conventional file) and instantiate it in any VM "host", independently of the location or the software configuration of the host. Furthermore, we can migrate running VMs to appropriate resources.

In the following, we begin by laying out the case for grid computing on virtual machines (Section 2), summarizing their advantages and showing that they come at little performance cost. Next, we describe the middleware challenges of our approach and explain how we are addressing them (Section 3). This is followed by a brief discussion of the grid computing architecture that we are designing (Section 4), related work (Section 5), and our conclusions (Section 6).

2 Why Grid Computing with Classic VMs?

The high-level answer to this question is that classic virtual machines provide a new abstraction layer, with very low overhead, that offers functionality that greatly simplifies addressing many of the issues of grid computing.

2.1 Definitions

A modern operating system uses multiprogramming, virtual memory, and file systems to share CPU, memory, and disk resources among multiple processes and users. Each process accesses the physical resources indirectly, through abstractions provided by the operating system. Contemporaneous to the development of these mechanisms was that of another resource-sharing approach, virtual machines [23]. The abstraction of a virtual machine is that of the underlying physical machine—each user appears to have a dedicated machine at their disposal, the hardware of which they can access directly.

Virtual machines can be divided into two main categories [27]: those that support a complete instruction set architecture (ISA-VMs) including both user and system instructions, and those that support an application binary interface (ABI-VMs) with user instructions and system/library calls. Same-ISA virtual machines typically achieve better performance than different-ISA VMs since they support native instruction execution without requiring binary modifications or run-time translations. An important class of virtual machines ("classic" VMs) consists of ISA-VMs that supports same-ISA execution of entire operating systems (e.g. the commercial products from the IBM S/390 series [18] and VMware [29], and the open-source project plex86 [21]). It is classic virtual machines that we target.

2.2 Advantages

Unlike conventional operating systems, classic VMs allow efficient dynamic multiplexing of users onto physical resources at the granularity of a single user per operating system session, thereby supporting per-user VM configuration and isolation from other users sharing the same physical resource. In the remainder of this section we focus on a scenario where each dynamic instance of a classic VM is dedicated to a single logical user. [1]

Security: The ability to share resources is a basic requirement for the deployment of grids; the integrity and security of shared resources is therefore a prime concern. A security model where resource providers trust the integrity of user codes restricts the application of grids to cases where mutual trust can be established between providers and users. If users are to submit jobs to computational grids without such trust relationship, the

[1] As depicted in Figure 5, it is possible to map a logical user to a single physical user, as well as to use grid middleware to multiplex a logical user across several physical users or applications, such as in PUNCH [20].

integrity of a computation may be compromised by a malicious resource [32], and, conversely, the integrity of the resource may be compromised by a malicious user [7].

Classic VMs achieve stronger software security than a conventional multiprogrammed operating system approach if redundant and independent mechanisms are implemented across the virtual machine monitor (VMM) and the operating system [22]. In a scenario where grid users have access to classic VMs, it is more difficult for a malicious user to compromise the resource than in conventional multiprogrammed OSes, because they must be able to break two levels of security: the VMM and the host OS.

Isolation: When more than one user uses a computational resource, classic VMs allow for stronger isolation among users, in addition to stronger resource security. In our scenario, a malicious user must be able to (a) break out from its VMM, (b) break security mechanisms of the host OS to obtain access to another user's VMM, and (c) break into another user's VMM.

Customization: Virtual machines can be highly customized without requiring system restarts. It is possible to specify virtual hardware parameters, such as memory and disk sizes, as well as system software parameters, such as operating system version and kernel configuration. Furthermore, multiple independent OSes can co-exist in the same server hardware. In a grid environment it becomes possible to offer virtual machines that satisfy individual user requirements from a pool of standard (physical) machines.

Legacy support: Virtual machines support compatibility at the level of binary code: no re-compilation or dynamic re-linking is necessary to port a legacy application to a VM. Furthermore, the legacy support provided by classic VMs is not restricted to *applications*: entire legacy *environments*—virtual hardware, the operating system, and applications—are possible.

Administrator privileges: In typical shared multiprogrammed systems, sensitive system operations are reserved to a privileged user—the system administrator. These operations are restricted to a trusted entity because they can compromise the integrity of the resource and/or of other users. In many situations, however, the need to protect system integrity forces a conservative approach in determining which operations are privileged, at the expense of possibly limiting forms of legitimate usage of the system. For example, the "mount" command is typically privileged, thus not accessible by common users. This prevents malicious users from gaining unauthorized access to local resources, but also disallows legitimate-use cases: e.g. a user who wishes to access remote data from an NFS partition setup at his or her computer at home.

When classic VMs are deployed under the assumption that each (logical) user has a dedicated machine, these requirements can be relaxed. The integrity of the resource underlying the OS (i.e. the virtual machine) is independent from the integrity of the multiplexed computer (i.e. the physical machine). Further, there are no users sharing the virtual machine. If necessary it is then possible to grant “root” privileges to untrusted grid users because the actions of malicious users are confined to their VMs.

Resource control: Some of the resources used by a classic VM (e.g. memory and disk sizes) can be customized dynamically at instantiation time. It is also possible to implement mechanisms to limit the amount of resources utilized by a VM at run-time by implementing scheduling policies at the level of the virtual machine monitor.

Unlike typical multi-programming environments, where resource control mechanisms are applied on a per-process basis, classic VMs allow complementary resource control at a coarser granularity—that of the collection of resources accessed by a user. Furthermore, resource control policies can be established dynamically. Dynamic resource control is important in a grid environment for two key reasons. First, it allows a provider to limit the impact that a remote user may have on resources available for a local user (e.g. in a desktop executing interactive applications). Second, it enables a provider to account for the usage of a resource (e.g. in a CPU-server environment). Resource control mechanisms based on classic VMs are particularly important in a grid environment since, unlike Java-oriented solutions [30], they can be applied to legacy application binaries. Section 3.2 elaborates on resource management issues that arise in in this scenario.

Site-independence: Classic VMs allow computation to be decoupled from idiosyncrasies of the site that hosts a physical machine. A VM guest presents a consistent run-time software environment—regardless of the software configuration of the VM host. This capability is very important in a grid environment: combined with the strong security and isolation properties of classic VMs, it enables cross-domain scheduling of *entire* computation environments (including OS, processes, and memory/disk contents of a VM guest) in a manner that is decoupled from site-specific administration policies implemented in the VM hosts.

A virtual machine can be instantiated on any resources that are sufficiently powerful to support it because it is not tied to particular physical resources. Furthermore, a running virtual machine can be suspended and resumed, providing a mechanism to migrate a running machine from resource to resource.

2.3 Performance considerations

The advantages of virtual machines are for naught if they can not deliver sufficient performance. Earlier work has shown that the impact of a particular virtual machine monitor, VMware, on CPU and network performance is relatively low [29]. In this section we report on measurements that show the overhead to be low for CPU-intensive tasks (less than 10%, for micro and macro benchmarks). The experimental data also shows that the costs of instantiating a dynamic virtual machine instance can be quite low, on the order of seconds.

Figure 1 summarizes the results of experiments using a microbenchmark intended to evaluate the degree

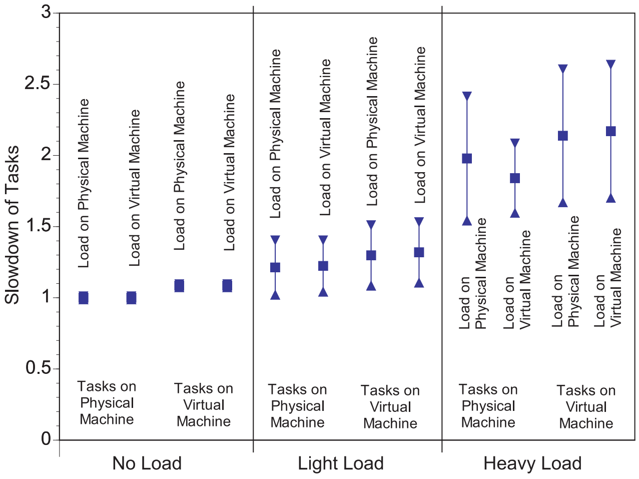

Figure 1. Microbenchmark results.

to which a VMware-based VM monitor slows down a compute-intensive task in the presence of background load. The compute node is a dual Pentium III/800MHz node with 1GB memory running RedHat 7.1. The virtual machine uses VMware Workstation 3.0a, with 128MB of memory, 2GB virtual disk and RedHat 7.2. The background load was produced by host load trace playback [12] of load traces collected on the Pittsburgh Supercomputing Center's Alpha Cluster. Three types of background load are used: none, light and heavy. In each case, we look at all four possible combinations of placing load and test tasks (those whose slowdown we measure) on the physical machine and the virtual machine. In the figure, we show the average slowdown of 1000 samples and the +/- one standard deviation. The main takeaway is that, independently of load (which generates context switches that preempt the VMM), the test tasks see almost identical slowdown when running on the virtual machine case as when running on the physical machine case. The average VM overhead is typically below 10%.

The low VM overhead holds true in large applications as well. Figure 2 shows the results for two macrobenchmarks. We executed the SPEChpc benchmarks SPECseis and SPECclimate on physical hardware and on a virtual machine. The benchmarks are compiled with OmniCC 1.4 (front-end) and gcc 2.96 (back-end), and executed in sequential mode. The compute node is a dual Pentium III/933MHz node with 512MB memory running RedHat 7.1. The virtual machine uses VMware Workstation 3.0a, with 128MB of memory, 1GB virtual disk and RedHat 7.1. The execution time of the benchmarks running on a VMware/x86-based virtual machine is within 4% of the native execution time.

The more quickly we can instantiate a virtual machine, the more widely this abstraction can be used in grid computing. We have conducted experiments that show the overhead of dynamically instantiating a VM using existing grid-based job submission mechanisms. In this experiment the processor, memory and disk state of the VM are accessible from the host OS as regular files, and the VMM (VMWare) is instantiated as a regular

UNIX process. The guest OS is Red Hat Linux. Two possible ways of instantiating VMs are considered:

- VM-reboot: the VM's guest OS is booted upon intialization
- VM-restore: the VM's guest OS is restored to a post-boot "warm" state.

Orthogonally, two different forms of storing the VM state files are considered:

- DSK-diff: the VM's disk is non-persistent; the disk is not explicitly copied upon startup, and modifications are stored into a diff file. Two forms of access to the disk are considered:
    - DIFF-native: state is stored in the local disk (native file system) of the host
    - DIFF-loopback: state resides in a loopback-mounted NFS partition of the host, simulating a remote file system.
- DSK-replica: an explicit copy of a persistent disk is created in the native file system of the host before the VM starts up.

The results of our experiment are shown in Figure 3. In the best case we studied, we could instantiate a virtual machine in about 12 seconds on average. This was achieved via a restoration using a non-persistent disk and difference file on the native file system. The start-up overhead increases to more than 4 minutes if explicit copies of a VM disk need to be generated, while remaining below 30 seconds if the VM state is accessed via a low-latency NFS/RPC stack.

3 Middleware challenges

Virtual machines provide a powerful new layer of abstraction in distributed computing environments. Since virtual machine monitors are readily available, it is certainly possible to deploy VMs as static com-

Application	**Resource**	**User time**	**Sys time**	**User+sys**	**Overhead**
SPECseis	Physical	16395s	19s	16414s	N/A
	VM, local disk	16557s	60s	16617s	1.2%
	VM, PVFS	16601s	149s	16750s	2.0%
SPECclimate	Physical	9304s	3s	9307s	N/A
	VM, local disk	9679s	5s	9679s	4.0%
	VM, PVFS	9695s	7s	9702s	4.2%

Figure 2. Macrobenchmark results. User, system and total times are reported for three scenarios: physical machine, VM with state in local disk, VM with state accessed via NFS-based grid virtual file system (PVFS). Overheads are calculated using execution times and the physical machine as reference. In the PVFS scenario, the physical and data servers are located at Northwestern University, while the image server is located at the University of Florida.

	VM-reboot			VM-restore		
	DSK-replica	DSK-diff		DSK-replica	DSK-diff	
		DIFF-native	DIFF-loopback		DIFF-native	DIFF-loopback
Mean	273	69.2	74.5	269	12.4	29.2
Std	21	6.9	2.0	17	4.6	7.0
Min	232	64.3	72.8	234	9.6	23.0
Max	304	86.3	79.8	302	24.9	44.2

Figure 3. Average, standard deviation, minimum and maximum VM startup times. Virtual machine sessions based on the configuration of Figure 1 are instantiated using globusrun (Globus 2.0 toolkit). Measurements have been taken across 10 samples. Time (in seconds) is measured as wall-clock execution time from the beginning to the end of the execution of globusrun.

puting units with existing grid middleware running within them. However, this new abstraction layer is only fully exploited when VMs are instantiated and managed *dynamically*. This section outlines the challenges and possible techniques to enable a dynamic virtual computing model and its integration with existing grid middleware solutions.

3.1 Data management

Data management is a key technology for VM-based grid computing, enabling administrative decoupling of computation providers and users. Data management involves: the transfer of VM images so that a user's virtual machine can be instantiated anywhere and migrated when necessary, and support for location-independent access to user files. With appropriate data management support, computation, state, and user data can reside in different domains.

The components of a virtual machine session are distributed across three different logical entities: *image servers*, which provide the capability of archiving static VM states; *computation servers* (or VM hosts), which provide the capability of instantiating dynamic VM images (or VM guests); and *data servers*, which provide the capability of storing user data. In this scenario, VM state information needs to be transferred from an image server to a VM host (where it is instantiated), and from a data server to the VM guest (where it is processed) as in Figure 4.

High performance data transfers: Fast and simple access to images and user data is critical. Current grid solutions, such as Globus [4, 1] and PBS [3] typically employ file-staging techniques to transfer files between user accounts in the absence of a common file system. File staging approaches require the user to specify the files to be transferred, transfer whole files when they are opened, and pose application programming

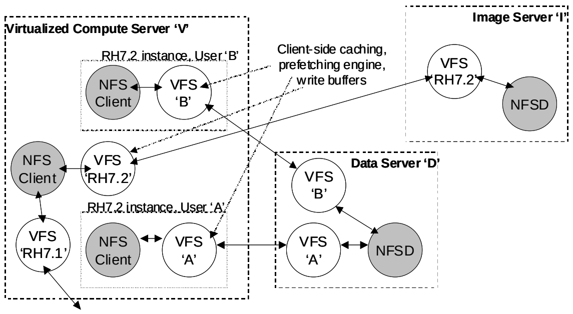

Figure 4. VM image and data management via virtual file systems. Users A and B are multiplexed onto the server V via two instances of Red Hat 7.2 virtual machines. Client-side VFS proxies at the host V cache VM state from image servers (e.g. server I), while proxies within virtual machines cache user blocks from a data server D.

challenges. Data management solutions that support on-demand transfer have also been deployed within Condor [31] and Legion [34].

Within the context of the PUNCH virtual file system (PVFS), previous work has shown that a data management model supporting simple on-demand data transfers without requiring dynamically-linked libraries or changes to native OS file system clients and servers can be achieved by way of two mechanisms: logical user accounts [19] and a virtual file system [14]. PVFS supports on-demand block transfers with performance within 1% of the underlying NFS file system [14]. Virtual machines naturally support a logical user account abstraction because dedicated VM guests can be assigned on a per-user basis, and the user identities within a VM guest are completely decoupled from the identities of its VM host. Furthermore, virtual machines provide an environment where legacy applications and OSes can be deployed—including services such as virtual file systems. In other words, VMs provide a layer of abstraction that supports logical users and virtual file systems (Figure 4). We can thus use these mechanisms for high performance access to images and user data.

We are also developing and evaluating novel on-demand data transfer solutions that improve upon the performance of block-based virtual file systems, and comparing them to existing static data transfer models for managing VM states and user data.

Image management: The state associated with a static VM image is usually larger than the working set that is associated with a dynamic VM instance. The transfer of entire VM states can lead to unnecessary traffic due to the copying of unused data [25]. On-demand transfers are therefore desirable. In addition, in the common case, large parts of VM images can shared by multiple readers (e.g. a master static Linux virtual system disk can be shared by multiple dynamic instances, as in Figure 4).

We are developing mechanisms to exploit such read-only sharing patterns within proxy-based virtual file systems. These include predicting the access patterns to VM state (disk, memory) based on previous history collected by file system proxies, and using this information to guide client-side caching and prefetching of file blocks to support efficient, on-demand transfers of VM images [2].

User and application data management: Several techniques exist for the transfer of user and application data. We are investigating the proxy-based virtual file system approach for efficient, location-transparent, on-demand access to user and application data. Unlike images, however, file system sessions for data management can be initiated within a VM guest (Figure 4).

Virtual machine migration: Combining image management, user and application data management, and checkpointing, a VM-based grid deployment can support the seamless migration of entire computing environments to different virtualized compute servers while keeping remote data connections active.

3.2 Resource management

Virtual machines provide a powerful new layer of abstraction in distributed computing environments, one that creates new opportunities and challenges for scheduling and resource management. Intriguingly, this is true both from the perspective of resources "looking up" at applications and applications "looking down" at resources.

Resource perspective: From the perspective of computational and communications resources "looking up" at applications, virtual machines provide a mechanism for carefully controlling how and when the resources are used. This is important because resource owners are far more likely to allow others to use the resources, or sell access to them, if they have such control. While there are other mechanisms for providing such fine-grain control, they impose particular systems software interfaces or computational models [26, 6, 24] on the user. Virtual machines, on the other hand, are straightforward—the user gets a ”raw” machine on which he/she can run whatever he pleases. The resource owner in turn sees a single entity to schedule onto his/her resources. How do we schedule a virtual machine onto the actual physical resources in order to meet the owner's constraints?

Our approach to the complex and varying constraints of resource owners is to use a specialized language for specifying the constraints, and to use a toolchain for enforcing constraints specified in the language when scheduling virtual machines on the host operating system. The basic idea is to take the resource owner's constraints and the constraints of the virtual machines that the users require and compile them into a real-time schedule, mapping each virtual machine into one or more periodic real-time tasks on the underlying host operating system. The complete real-time schedule is such that the owner's constraints are not violated.

[2] Including transfers from, and to, the image server, to support not only instantiation of a VM's state in remote resources, but also checkpointing.

We currently target Real-time Linux [35], KURT [28], or RTAI [5]. We also plan to investigate other techniques, including modulating the priority of virtual machine processes under the regular linux scheduler, using SIGSTOP/SIGCONT signal delivery to virtual machine processes to approximate the desired real-time schedule, and implementing extensions to the Linux scheduler.

Application perspective: To achieve appropriate performance on distributed computing environments, applications typically have to adapt to the static and dynamic properties of the available resources. Virtual machines make this process simpler in some respects by allowing the application to bring its preferred execution environment along with it. However, complexity is introduced in other respects. First, virtual machines are themselves a new resource, increasing the pool of resources to be considered. Second, virtual machines represent collections of shares in the underlying physical resources. To predict its performance on a particular virtual machine or group of virtual machines, the application must understand the mapping and scheduling of virtual resources onto the underlying physical resources, or there must be some service that does this for it.

For static properties of virtual resources, we are currently extending an existing relational database approach for capturing and querying the static properties of resources with a computational grid [10]. The basic idea is that applications can best discover a collection of appropriate resources by posing a relational query including joins. In our model, such queries are nondeterministic and return partial results in a bounded amount of time. We are extending the model to include virtual machines. Virtual machines would register when instantiated. Hosts would advertise what kinds and how many virtual machines they were willing to instantiate (virtual machine futures). The service would also contain information about how the virtual machines are scheduled to the underlying hardware, information derived from the constraints-to-schedule compilation process described above. Applications would be able to query over virtual machines or virtual machine futures.

Applications typically must also adapt to dynamic changes in resource supply. The RPS system [11] is designed to help this form of adaptation. Fed by a streaming time-series produced by a resource sensor, it provides time-series and application-level performance predictions on which basis applications can make adaptation decisions. Currently, RPS includes sensors for Unix host load, network bandwidth along flows in the network, Windows performance counters, and can be extended to include sensors that are appropriate for VM environments.

3.3 Virtual networking

While a virtual machine monitor such as VMWare can create a virtual machine, that machine must be able to connect to a network accessible by a computational grid. Unlike a process running on the underlying physical machine, the virtual machine appears to the network to be one or more new network interface cards. The integration of a dynamically created VM to the network is dependent upon the policies implemented in the site hosts the (physical) VM server. With respect to these policies, two scenarios can arise.

1. The VM host has provisions for IP addresses that can be given out to dynamic VM instances. For instance, a CPU farm may provide the capability of instantiating full-blown virtual back-ends as a service (as in Figure 5). In this scenario, the VM may obtain an IP address dynamically from the host's network (e.g. via DHCP), which can then be used by the middleware to reference the VM for the duration of a session.

2. The VM host does not provide IP addresses to VM instances. In this scenario, network virtualization techniques — similar to VPNs [13] — may be applied to assign a network identity to the VM at the user's (client) site. The simplest approach is to tunnel traffic, at the Ethernet level, between the remote virtual machine and the local network of the user. In this way, the remote machine would appear to be connected to the local network, where, presumably, it would be easy for the user to have it assigned an address, etc. If we can establish a TCP connection to the remote site, which we must in order to launch the virtual machine in the first place, we will be able to use it for tunneling. For example, if we used SSH to start the machine, we could use the SSH tunneling features. A natural extension to this simple VPN in which all remote hosts appear on the local network is to establish an overlay network among the remote virtual machines [2]. The overlay network would optimize itself with respect to the communication between the virtual machines and the limitations of the various sites on which they run.

3.4 Integration with existing Grid infrastructures

The VM-based mechanisms described in this paper allow seamless integration of virtualized end-resources with existing and future Grid-based services. This integration can be achieved at the level of grid middleware, and can leverage mechanisms from open-standard Grid software, such as the Globus toolkit [16].

This integration is based on the convenient property that entire VM environments can be regarded as a combination of traditional OS processes (the VM monitors) and files (the VM state). Using this abstraction, traditional information services (e.g. MDS [15], URGIS [10]) can be used to represent VMs as Grid resources; resource management services (e.g. GRAM [8]) can be used to dispatch VM environments; and data management services (e.g. GASS [4], GridFTP [1] and Grid virtual file systems [14]) can be used to handle the transfer of virtual machine state and application data.

4 Architecture

In the following we lay out an initial software architecture for virtual machine grid computing by describing the life cycle of a VM within it.

In this architecture, the nodes of a virtual computational grid support, in addition to virtual machine monitors, a set of tools that limit the share of resources that the virtual machines are permitted to use, grid

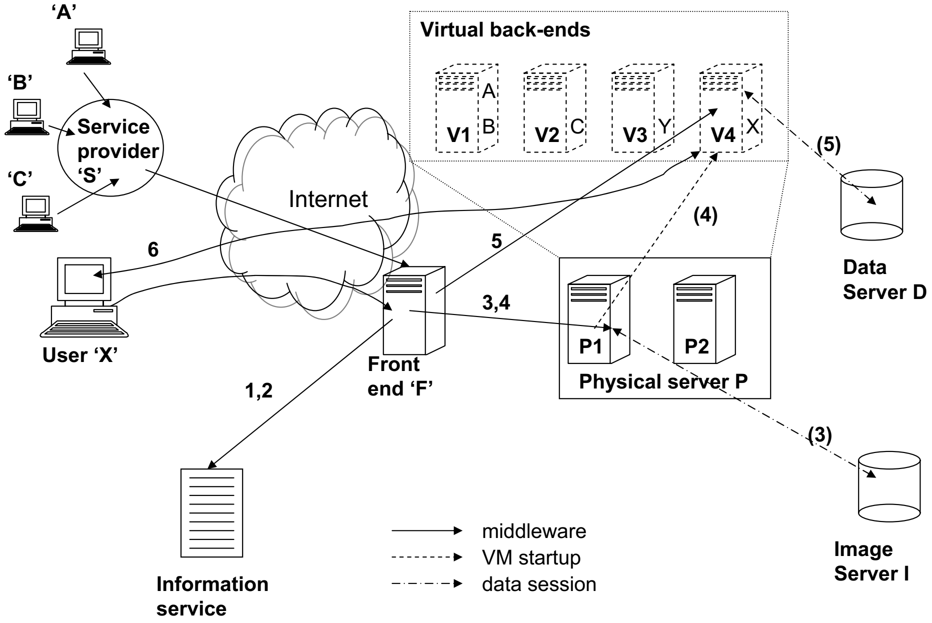

Figure 5. Architecture for a VM-based grid service. In 1-6, a virtual machine (V4) is dynamically created by middleware front-end F on behalf of user X. This VM is decidated to a single user. In another scenario, virtual machines V1, V2 are instantiated on P2 on behalf of a service provider S, and are multiplexed across users A, B, C and applications provided by S. The logical user account abstraction decouples access to physical resources (middleware) from access to virtual resources (end-users and services).

middleware such as Globus (and SSH) for instantiating machines, and resource monitoring software such as RPS. Virtual machine instances or the capability for instantiating virtual machines (virtual machine futures) will be advertised via a grid information service such as Globus MDS or URGIS. Virtual file systems will give all nodes access to currently stored VM images. User accounts, implemented as Globus accounts or SSH keys, will allow users only to instantiate and store virtual machines.

In the discussion to follow, we consider a generic scenario where the components of a virtual grid session — physical server, virtual machine O/S image server, application image server, and user data server — are distributed across nodes of a grid. The steps taken by the VM-based grid architecture to establish a virtual machine session for a user are as follows (refer to Figure 5):

1. A user **X** (or grid middleware **F** on their behalf) first consults an information service, querying for a VM future (a physical machine able to instantiate a dynamic VM) **P** that meets their needs.

2. If necessary, **X** also consults an information service to query for a VM image server **I** with a base O/S installation that meet their application needs. Alternatively, users may provide VM images of their own (e.g. a customized O/S installation).

3. The middleware then establishes a data session between the physical server **P** and the image server **I** to allow for the instantiation of a dynamic VM. This data connection can be established via explicit transfers (e.g. GridFTP) or via implicit, on-demand transfers (e.g. a grid virtual file system, Figure 4).

4. Once the data session for image **I** is established, the user can negotiate with the physical machine the startup of a VM instance **Vi** (e.g. using Globus GRAM or SSH). The virtual machine **Vi** may start from a pre-boot (cold) state, or from a post-boot (warm) state stored as part of the image. In addition, upon startup, the VM is assigned an IP address (via DHCP, or by connecting to a virtual network).

5. Once the VM instance **Vi** is running and on the network, additional data sessions are established. These connect the O/S *within* **Vi** to application server **A** and to the user's data server **D**. As previously, these sessions can be realized with explicit or implicit transfers (Figure 4).

6. The application executes in the virtual machine; if it is an interactive application, a handle is provided back to the user (e.g. a login session, or a virtual display session such as VNC).

In this setup, the user can have the choice of whether to be presented with a console for the virtual machine (e.g. the application may the O/S console itself) or to run without this interface (e.g. for batch tasks). The user, or a grid scheduler, will have the option to shutdown, hibernate, restore, or migrate the virtual machine at any time. In large part, these processes will use the same mechanisms: efficient transfer of the current image, either to another machine or to a file, adjustments to the VPN, and continual connectivity to files via the virtual file system. Infrequently run virtual machine images will be migrated to tape. The life cycle of a virtual machine ends when the image is removed from permanent storage.

The data summarized in Figure 2 shows that a setup with distributed physical, image and data servers is feasible, from performance and implementation standpoints, for applications that are CPU-intensive. In this experiment, the on-demand data session between **P** and **I** was established via NFS-based PVFS proxies [14] across a wide-area network, and the connection between **V** and **D** was established via PVFS across two VMs in a local area network. The observed execution time overhead is small. This experiment considers a scenario where no locality-enhancement techniques (other than those implemented by kernel-level NFS components) are applied. As an enhancement, it is possible to seamlessly integrate proxy-level locality techniques into the architecture.

5 Related Work

“Classic” VMs have been used as a means of multiplexing shared mainframe resources since the early seventies. In the past years, the demands for computation outsourcing and resource consolidation has prompted the development of VM-based solutions that deliver commodity OSes from mainframes (e.g. Linux on IBM S/390) and microprocessor-based hardware (e.g. Linux/Windows on x86/VMware). We are seeking to leverage classic VMs in a new context, grid computing.

The Denali project [33] is similar to ours in that it has a similar objective of providing network-based services based on VMs. Denali focuses on supporting lightweight VMs, relying on modifications to the virtual instruction set exposed to the guest OS and thus requiring modifications to the guest OS. In contrast, we are focusing on heavier weight VMs and make no OS modifications. User-mode VMs have been recently proposed for the Linux OS [9]. Although this approach allows for user isolation, unlike classic VMs it does not support arbitrary guest OSes, which is a necessity for us. “Computing capsules” that can be dynamically instantiated as computation caches for arbitrary, legacy applications are being explored at Stanford [25]. However, this approach does not simultaneously multiplex different full-fledged OSes in a single host.

6 Conclusions

Classic virtual machines support a grid computing abstraction where computation becomes decoupled from the underlying physical resources. In this model, entire computing environments can be represented as *data* (a large state) and physical machines can be represented as *resources* for instantiating data. This abstraction is powerful because it decouples the administration of *computing users* from the administration of *resource providers*. This simplifies addressing many issues in grid computing and provides a new layer at which to work.

We have presented a qualitative argument for the use of virtual machines in grid computing and quantitative results that demonstrate the feasibility of this idea from a performance perspective. We then illustrated the middleware challenges that must be overcome to build grid computing on top of virtual machine monitors and layed out how we are addressing those challenges. Finally, we provided a description of our nascent software architecture and its integration with existing middleware to support a VM-based infrastructure for computational grids.

References

[1] B. Allcock, J. Bester, J. Breshanan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and Steven Tuecke. Secure, efficient data transport and replica management for high-performance data-intensive computing. In *IEEE Mass Storage Conference*, 2001.

[2] David G. Andersen, H. Balakrishnan, Frank Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of the 18th ACM SOSP*, Banff, Canada, October 2001.

[3] Albeaus Bayucan, Robert L. Henderson, Casimir Lesiak, Bhroam Mann, Tom Proett, and Dave Tweten. Portable Batch System: External reference specification. Technical report, MRJ Technology Solutions, November 1999.

[4] Joseph Bester, Ian Foster, Carl Kesselman, Jean Tedesco, and Steven Tuecke. GASS: A data movement and access service for wide area computing systems. In *Proceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems*, May 1999.

[5] E. Bianchi, L. Dozio, G.L. Ghiringhelli, and P. Mantegazza. Complex control systems, applications of DIAPM-RTAI at DIAPM. In *Proceedings of the First Real Time Linux Workshop*, 1999.

[6] Greg Bollella, Ben Brogsol, Peter Dibble, Steve Furr, James Gosling, David Hardin, Mark Turnbull, Rudy Belliardi, Doug Locke, Scott Robbins, Pratik Solanki, and Dionisio de Niz. The real-time specification for java. Addison-Wesley, November 2001. http://www.rtj.org/rtsj-V1.0.pdf.

[7] Ali R. Butt, Sumalatha Adabala, Nirav H. Kapadia, Renato J. Figueiredo, and José A. B. Fortes. Fine-grain access control for securing shared resources in computational grids. In *To appear, Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, April 2002.

[8] Karl Czajkowski, Ian Foster, N. Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Teucke. A resource management architecture for metacomputing systems. In *Proceedings of the Fourth Workshop on Job Scheduling Strategies for Parallel Processing*, 1998. Held in conjunction with the International Parallel and Distributed Processing Symposium.

[9] J. Dike. A user-mode port of the linux kernel. In *Proceedings of the USENIX Annual Linux Showcase and Conference*, Atlanta, GA, Oct 2000.

[10] Peter Dinda and Beth Plale. A unified relational approach to grid information services. Technical Report GWD-GIS-012-1, Global Grid Forum, 2001.

[11] Peter A. Dinda and David R. O'Hallaron. An extensible toolkit for resource prediction in distributed systems. Technical Report CMU-CS-99-138, School of Computer Science, Carnegie Mellon University, July 1999.

[12] Peter A. Dinda and David R. O'Hallaron. Realistic CPU workloads through host load trace playback. In *Proc. of 5th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR2000)*, volume 1915 of *Lecture Notes in Computer Science*, Rochester, New York, May 2000. Springer-Verlag.

[13] Nick G. Duffield, Pawan Goyal, Albert G. Greenberg, Partho Pratim Mishra, K. K. Ramakrishnan, and Jacobus E. van der Merive. A flexible model for resource management in virtual private networks. In *SIGCOMM*, pages 95–108, 1999.

[14] Renato J. Figueiredo, Nirav H. Kapadia, and José A. B. Fortes. The PUNCH virtual file system: Seamless access to decentralized storage services in a computational grid. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC'01)*, San Francisco, California, August 2001.

[15] Steven Fitzgerald, Ian Foster, Carl Kesselman, Gregor von Laszewski, Warren Smith, and Steven Tuecke. A directory service for configuring high-performance distributed computations. In *Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing (HPDC'97)*, pages 365–375, 1997.

[16] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2), 1997.

[17] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[18] IBM Corporation. White paper: S/390 virtual image facility for linux, guide and reference. GC24-5930-03, Feb 2001.

[19] Nirav H. Kapadia, Renato J. O. Figueiredo, and José A. B. Fortes. Enhancing the scalability and usability of computational grids via logical user accounts and virtual file systems. In *Proceedings of the Heterogeneous Computing Workshop (HCW) at the International Parallel and Distributed Processing Symposium (IPDPS)*, San Francisco, California, April 2001.

[20] Nirav H. Kapadia and José A. B. Fortes. On the design of a demand-based network-computing system: The Purdue University Network-Computing Hubs. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC'98)*, pages 71–80, Chicago, Illinois, July 1998.

[21] Kevin Lawton. Running multiple operating systems concurrently on an ia32 pc using virtualization techniques. www.plex86.org/research/paper.txt.

[22] Stuart E. Madnick and John J. Donovan. Application and analysis of the virtual machine approach to information system security and isolation. In *Proc. ACM SIGARCH-SYSOPS Workshop on Virtual Computer Systems*, pages 210–224, Boston, MA, March 1973.

[23] R. A. Meyer and L. H. Seawright. A virtual machine time sharing system. *IBM System Journal*, 9(3):199–218, 1970.

[24] Raj Rajkumar, Kanaka Juvva, Anastasio Molano, and Shui Oikawa. Resource kernels: A resource-centric approach to real-time systems. In *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, January 1998.

[25] B. K. Schmidt. *Supporting Ubiquitous Computing with Stateless Consoles and Computation Caches*. PhD thesis, Department of CS, Stanford University, Aug 2000.

[26] Douglas C. Schmidt, Aniruddha Gokhale, Timothy H. Harrison, and Guru Parulkar. A high-performance endsystem architecture for real-time CORBA. *IEEE Communication Magazine*, 14(2), February 1997.

[27] James E. Smith. An overview of virtual machine architectures. http://www.ece.wisc.edu/ jes/papers/vms.pdf, Oct 2001.

[28] Balaji Srinivasan, Shyamalan Pather, Robert Hill, Furquan Ansari, and Douglas Niehaus. A firm real-time system implementation using commerical off-the-shelf hardware and free software. In *Proceedings of the IEEE Real-time Technology Applications Symposium (RTAS 1998)*, pages 112–120, 1998.

[29] Jeremy Sugerman, Ganesh Venkitachalan, and Beng-Hong Lim. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In *Proceedings of the USENIX Annual Technical Conference*, June 2001.

[30] Niranjan Suri, Jeffrey M. Bradshaw, Maggie R. Breedy, Kenneth M. Ford, Paul T. Groth, Gregory A. Hill, and Raul Saavedra. State capture and resource control for java: The design and implementation of the aroma virtual machine. In *Java Virtual Machine Research and Technology Symposium, USENIX*, April 2001.

[31] Douglas Thain, Jim Basney, Se-Chang Son, and Miron Livny. The kangaroo approach to data movement on the grid. In *Proceedings of the 2001 IEEE International Conference on High-Performance Distributed Computing (HPDC)*, pages 325–333, Aug. 2001.

[32] G. Vigna, editor. *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*. Springer-Verlag, June 1998.

[33] A. Whitaker, M. Shaw, and S. Gribble. Denali: Lightweight virtual machines for distributed and networked applications. In *Proceedings of the USENIX Technical Conference*, Monterey, CA, June 2002.

[34] Brian S. White, Andrew S. Grimshaw, and Anh Nguyen-Tuong. Grid-based file access: The Legion I/O model. In *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC'00)*, pages 165–173, Pittsburgh, Pennsylvania, August 2000.

[35] V. Yodaiken and M. Barabanov. A real-time linux. Presented at USENIX 97; online at http://rtlinux.cs.nmt.edu/rtlinx/u.pdf.