Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure

Paul Ruth, Junghwan Rhee, Dongyan Xu
Department of Computer Science
Purdue University
West Lafayette, IN 47907, USA
{*ruth, rhee, dxu*}*@cs.purdue.edu*

Rick Kennell, Sebastien Goasguen
Rosen Center for Advanced Computing
Purdue University
West Lafayette, IN 47907, USA
{*linux, sebgoa*}*@purdue.edu*

Abstract

A shared distributed infrastructure is formed by federating computation resources from multiple domains. Such shared infrastructures are increasing in popularity and are providing massive amounts of aggregated computation resources to large numbers of users. Meanwhile, virtualization technologies, at machine and network levels, are maturing and enabling mutually isolated virtual computation environments for executing arbitrary parallel/distributed applications on top of such a shared physical infrastructure. In this paper, we go one step further by supporting autonomic adaptation of virtual computation environments as active, integrated entities. More specifically, driven by both dynamic availability of infrastructure resources and dynamic application resource demand, a virtual computation environment is able to automatically relocate itself across the infrastructure and scale its share of infrastructural resources. Such autonomic adaptation is transparent to both users of virtual environments and administrators of infrastructures, maintaining the look and feel of a stable, dedicated environment for the user. As our proof-of-concept, we present the design, implementation, and evaluation of a system called VIOLIN, which is composed of a virtual network of virtual machines capable of live migration across a multi-domain physical infrastructure.

1 Introduction

We have seen the emergence of shared distributed infrastructures that federate, allocate, and manage heterogeneous resources across multiple network domains, most notably PlanetLab [2] and the Grid [9, 8]. The growth of these infrastructures has led to the availability of unprecedented computational power to a large community of users. Meanwhile, virtual machine technology [1, 5, 20] has been increasingly adopted on top of such shared physical infrastructures [6], and has greatly elevated customization, isolation, and administrator privilege for users running applications inside individual virtual machines.

Going beyond individual virtual machines, our previous work proposed techniques for the creation of virtual distributed computation environments [10, 14, 15] on top of a shared distributed infrastructure. Our virtual computation environment, called a VIOLIN, is composed of virtual machines connected by a virtual network, which provides a layer separating the ownership, configuration, and administration of the VIOLIN from those of the underlying infrastructure. Mutually isolated VIOLINs can be created for different users as their "own" private distributed computation environment bearing the same look and feel of customized physical environments with administrative privilege (e.g., their own private cluster). Within VIOLIN, the user is able to execute and interact with unmodified parallel/distributed applications, and can expect strong confinement of potentially untrusted applications.

It is possible to realize VIOLIN environments as integrated, autonomic entities that dynamically adapt and relocate themselves for better performance of the applications running inside. This all software virtualization of distributed computation environments presents a unique opportunity to advance the possibilities of autonomic computing [21, 18]. The autonomic adaptation of virtual computation environments is driven by two main factors: (1) the dynamic, heterogeneous availability of infrastructure resources and (2) the dynamic resource

needs of the applications running inside VIOLIN environments. Dynamic resource availability may cause the VIOLIN environment to relocate its virtual machines to new physical hosts when current physical hosts experience increased workloads. At the same time, dynamic applications may require different amounts of resources throughout their execution. The changing requirements can trigger the VIOLIN to adapt its resource capacity in response to the application's needs. Furthermore, the autonomic adaptation (including relocation) of the virtual computation environment is *transparent* to the application and the user, giving the latter the illusion of a well-provisioned, private, networked run-time environment. To realize the vision of autonomic virtual environments we address the following challenges:

First, we must provide the mechanisms for application-transparent virtual environment adaptation. In order to provide a consistent environment, adaptation must occur without affecting the application or the user. Currently, work has be done to enable resource reallocation and migration within a local-area network [4] and most current machine virtualization platforms support migration. However, we still need to determine how to migrate virtual machines across a multi-domain environment without affecting the application. The solution must keep the virtual machine alive throughout the migration. Computation must continue and network connections must remain open. The necessary cross-domain migration facility requires two features not yet provided by current virtualization techniques. First, virtual machines need to retain the same IP addresses and remain accessible through the network when physical routers will not know where they were migrated. Second, cross-domain migration cannot rely on NFS to maintain a consistent view of the large virtual machine image files. These files must be transferred quickly across the network. Clearly, current solutions are not yet adequate for multi-domain.

The second challenge is to define *allocation policies*. Our goal is to move beyond the limits of static allocation and provide autonomic environments that have the intelligence to scale resource allocations without user intervention. As such, we need to determine when a virtual machine needs more CPU, which virtual machine should be migrated, and where to migrate the virtual machine when a host can no longer support the memory demands of its guests. Consequently, we must be able to recognize the best destination could either be the one to which we can quickly migrate or one with a long migration time but more adequate resources.

The main contribution of this paper is the autonomic adaptation capabilities of VIOLIN environments. These environments retain the customization and isolation properties of existing static VIOLINs, however, they may be migrated to another host domain during run-time. In this way we can make efficient use of available resources across multiple domains.

We have built a prototype adaptive VIOLIN system using Xen [1] virtual machines and have deployed it over the nanoHUB (www.nanohub.org) infrastructure. The evaluation of the system shows that we are able to provide increased performance to several concurrently running virtual environments. To the best of our knowledge, this is the first demonstration of an autonomic adaptive virtual environment, using live application-transparent migration with real-world parallel applications.

The remainder of this paper is organized as follows: Section 2 describes the design of VIOLIN autonomic virtual environments, Section 3 presents their real-world deployment, Section 4 describes the experiments and presents performance results, Section 5 compares our study to related works, and Section 6 presents the paper's conclusions.

2 Autonomic Virtual Environments

In the VIOLIN system, each user is presented with an isolated virtual computation environment of virtual machines connected by a virtual network. From the user's point of view, a virtual computation environment is a private cluster of machines dedicated to that user. The user does not know where the virtual machines reside. On the other hand, the infrastructure sees the environments as dynamic entities that can move through the infrastructure utilizing as much or as little resources as needed.

The components of the VIOLIN system can be seen in Figure 1 and are described below:

- **Enabling Mechanisms**: The enabling mechanisms include the VIOLIN virtual environments as well as the *monitoring daemon* running on each physical host. The VIOLIN environments provide an interface to the user and applications, while the *monitoring daemons* monitor the CPU and memory on each host by querying the local *virtual machine monitor* (VMM) for resource availability and utilization levels. In addition, the monitors can manipulate the allocation of resources to local virtual machines.

- **Adaptation Manager**: The *adaptation manager* queries the *monitoring daemons* to form a global view of all host resources available as well as the utilization level of the allocated resources. With

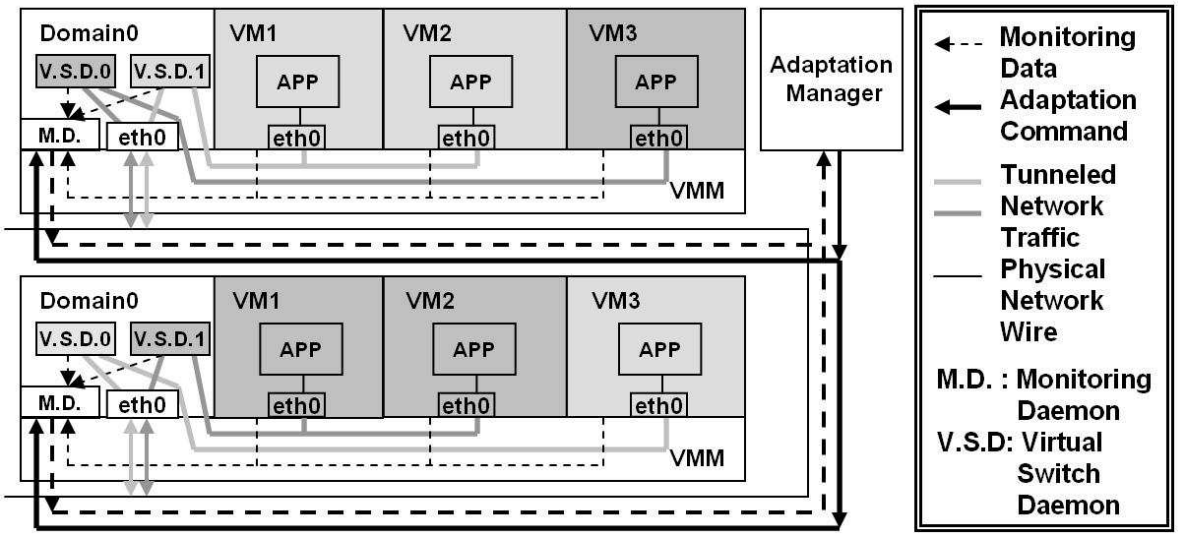

Figure 1. Multiple VIOLIN environments sharing two hosts. Daemons on each host assist the Adaptation Manager in monitoring and controlling resource allocation.

this information, the *adaptation manager* can dictate resource reallocation including fine-grained per-host CPU and memory adjustments, as well as coarse-grained migration of virtual machines or whole virtual environments without any user or administrator involvement.

2.1 Enabling Mechanisms

Local Adaptation Mechanism. The *adaptation manager* controls all virtual machines through the *monitoring daemons*. VIOLIN environments use both memory ballooning and weighted CPU scheduling to achieve fine-grained control over per-host memory and CPU allocation. Both VMware [20] and Xen [1] enable memory *ballooning* which allows the VMM to change the amount of memory allocated to each virtual machine while the machine is running. At run-time, the *adaptation manager* may decide to modify the memory footprint and percentage of CPU allocated through the monitoring daemons.

Multi-domain Adaptation Mechanism A key contribution of VIOLIN to autonomic adaptation is the ability to reallocate resources to virtual machines by migrating them live across networks. Live virtual machine migration is the transfer of a virtual machine from one host to another without pausing the virtual machine or checkpointing the applications running within the virtual machine. One of the major challenges of live migration is maintaining any network connections the virtual machine may have open. Modern machine virtualization mechanisms provide live virtual machine migration within layer-2 networks [4]. VIOLIN lifts this limitation by creating a virtual layer-2 network that tunnels network traffic end-to-end between remote virtual machines. The virtual network appears to be an isolated physical Ethernet LAN through which migration is possible. As the virtual machines move through the infrastructure, they will remain connected to their original virtual network.

2.2 Adaptation Manager

The *adaptation manager* is the intelligent agent, or "puppeteer" acting on behalf of the users and administrators and making autonomic reallocation decisions. It is appointed two tasks: to compile a global system-view of the available resources and to use this view to transparently adapt the allocation of global resources to virtual environments.

2.2.1 Infrastructure Resource Monitoring

The *adaptation manager* monitors the entire infrastructure by querying the *monitoring daemons* on each host. Via the monitors, it maintains knowledge of all available hosts in addition to the demands of applications running within the VIOLINs. Overtime both the

resources available in the shared infrastructure and the VIOLIN's utilization of resources will change. Hosts may be added or removed and VIOLINs can be created, destroyed, or enter periods of high or low CPU, memory, or network usage.

2.2.2 Resource Reallocation Policy

The *adaptation manager's* reallocation policy is based on observed host resource availability and virtual machine resource utilization. It uses a heuristic that aims to dynamically migrate overloaded virtual machines between hosts within each domain, and, if that is not possible, migrate overloaded VIOLINs between domains in the infrastructure. We do not attempt to find the optimal allocation of resources to virtual machines. Instead, we aim at incrementally increasing the performance of the system while minimizing the number of virtual machine migrations and the resulting overhead.

Intuitively, the policy determines a desired resource level for each virtual machine and attempts to assign that amount of resources to a virtual machine. If adequate resources cannot be obtained locally, the virtual machine may be migrated to another host or the whole VIOLIN may be migrated to another domain.

The *desired resource level* of each virtual machine is determined by the amount of allocated CPU and memory as well as the amount of resources that are actually being utilized. We wish to keep each virtual machine's resource utilization within a certain (predefined and configurable) range. A utilization level outside of the expected range will cause the *adaptation manager* to increase or decrease the virtual machine's resource allocation.

The heuristic finds over- and under-utilized virtual machines and attempts to adjust their allocations using first the local host's resources. If the local host cannot support all of its currently hosted virtual machines, an attempt is made to find another host within the domain to which one or more virtual machines can be migrated. The heuristic first looks at the hosts within the domain that have the lowest utilization level. If no hosts can support the over-utilized virtual machine, the whole domain is considered overloaded and an attempt is made to find another domain which can support the resource needs of one or more of the overloaded domain's VIOLINs. If a destination domain is found, VIOLINs will be migrated live to hosts in that domain.

3 Implementation

We have implemented an adaptive VIOLIN system prototype and have deployed the system on the nanoHUB's (www.nanohub.org) infrastructure. The nanoHUB is an e-Science infrastructure running online and on-demand Nanotechnology applications, and is our "living lab". Part of the nanoHUB allows students and researchers to execute computational Nanotechnology applications, including distributed and parallel simulations, through either a web-based GUI or a VNC desktop session. The unique property of the nanoHUB is that the back-end processing is heavily reliant on virtualization. Users of the nanoHUB may, unknowingly, be using VIOLIN environments that have the ability to adapt resource allocation to the changing needs of their simulations.

3.1 Deployment Details

Toward a full deployment, we have deployed multiple adaptive VIOLINs on the nanoHUB's multi-domain infrastructure on the campus of Purdue University.

Host Infrastructure. The virtual machines are hosted on two independent clusters on separate subnets. One cluster is composed of 24 Dell 1750s each with 2GB of RAM and two hyper-threaded Pentium 4 processors running at 3.06 GHz, while the other is 22 Dell 1425s each with 2GB of RAM and two hyper-threaded Pentium 4 processors running at 3.00 GHz. Both clusters support Xen 3.0 virtual machines and VIOLIN virtual networking.

Virtual Environment Configuration. Each virtual computation environment is composed of Xen virtual machines connected by a VIOLIN network. Amongst the virtual machines, one is a head node and the rest are compute nodes. The head node provides users with access to the VIOLIN environment and, as such, must remain statically located within its original host domain. However, all compute nodes are free to move throughout the infrastructure as they remain connected via the VIOLIN virtual network.

User accounts are managed by a shared Lightweight Directory Access Protocol (LDAP) server and users home directories are mounted to the local NFS server with the head node acting as a NAT router for the isolated compute nodes, giving a consistent system view to all virtual machines regardless of the physical locations of the virtual machines.

In order to migrate a virtual machine, the following must be transferred to the destination host: a snapshot

of the root file system image, a snapshot of the current memory, and the thread of control. Xen's live migration capability supports efficient transfer of the memory and thread of control. It performs an iterative process that reduces the amount of time the virtual machine is unavailable to an almost unnoticeable level. However, Xen does not support the migration of the root file system image. Xen assumes that the root file system is available on both the source and destination hosts -usually through NFS which can not safely be made available between multiple domains. The shared infrastructure is composed of independently administered domains which cannot safely share NFS servers. In order to perform multi-domain migrations, our prototype uses read-only root images that can be distributed without having to be updated. We do this by putting all system files that need to be written to in *tmpfs* filesystems. Since *tmpfs* file systems are resident in memory, Xen will migrate these files with the memory. Initially, we thought of this solution as a workaround to be fixed later, however, our experience has demonstrated that *tmpfs* can be a reasonable solution for a number of nanoHUB applications. In addition to using *tmpfs* for system files, users home directories are NFS-mounted through the virtual network to the nanoHUB server and do not need to be explicitly transferred.

4 Experiments

In this section, we present several experiments that show the feasibility of adaptive VIOLIN environments. First, we measure the overhead of live migration of VIOLIN environments, then we demonstrate application performance improvement due to autonomic live adaptation of VIOLINs sharing a multi-domain infrastructure. For all experiments we use the nanoHUB VIOLIN deployment, an *adaptation manager* employing the heuristic described in section 2.2.2, and the NEMO3D [12] parallel atomic particle simulation as the application running in the VIOLINs.

4.1 Migration Overhead

Objective. This experiment aims to find the overhead of migrating an entire VIOLIN that is actively running a resource intensive application (individual virtual machine migration overheads have been studied in [4]). The overhead of live VIOLIN migration includes the execution time lost due to the temporary down-time of the virtual machines during migration, the time needed to reconfigure the VIOLIN virtual network, and any lingering effects such as network slowdown caused by packet loss and the resulting TCP back-off.

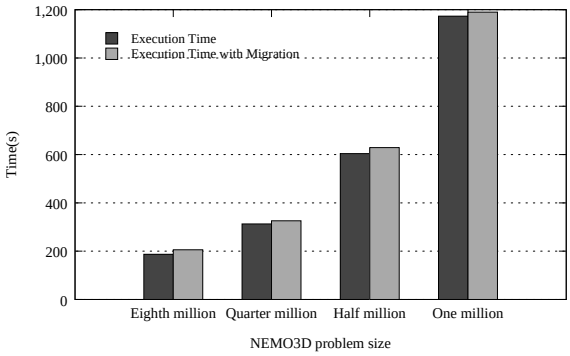

Figure 2. Migration overhead caused by live migration of entire VIOLIN virtual environments that are actively executing the parallel application NEMO3D

Configuration. We use a VIOLIN composed of four virtual machines. We execute NEMO3D with several different problem sizes between 1/8 and 1 million particles. For each problem size, we record the execution time with and without migrating the VIOLIN. During the no-migration runs, the application is allowed to run unimpeded. During each run involving migration, all four virtual machines are simultaneously migrated live across the network to destination hosts configured identically to the source hosts. In order to stress the system and find the worst overhead possible, we choose the migration to occur at the most resource intensive period of the application's execution. During each run, there is no background load in any of the hosts involved. However, the network is shared and therefore incurs background traffic.

Results. Figure 2 shows the results. We find that, regardless of problem size, the run-time of the application is increased by approximately 20 seconds (ranging from 17-25 seconds) when the VIOLIN is migrated.

Discussion. One requirement of adaptive VIOLIN environments is that there should be little or no effect on the applications due to adaptation. The 20 second penalty would seem impossible considering that Xen virtual machine migration requires the transfer of the entire memory footprint (approximately 800MB per virtual machine for an execution of NEMO3D simulating 1 million particles). However, Xen's live migration mechanism hides the migration latency by continuing to run the application in the virtual machine on the source host while the bulk of the memory is being transferred. We

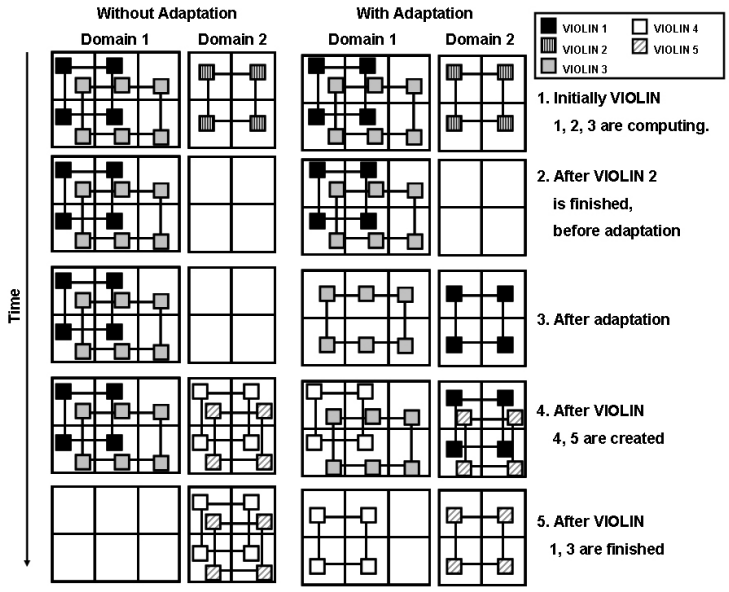

Figure 3. VIOLIN Adaptation Scenario 1.

do not measure the actual down-time of our virtual machines; however, Xen migration of a virtual machine with 800MB of memory was found to have a 165ms down-time when migrating within a LAN [4]. The major effect on application performance is not due to the migration itself but the time to reestablish the VIOLIN virtual network plus application slowdown *during* the migration. This experiment shows that the penalty for migrating a VIOLIN environment is relatively small and does not escalate with increased virtual machine memory size.

4.2 VIOLIN Adaptation Scenario 1

Objective. The purpose of this experiment is to demonstrate the effectiveness of the *adaptation manager* and to show how a small amount of autonomic adaptation can lead to better performance of all VIOLIN environments that share the infrastructure.

Configuration. We launch five VIOLIN environments, each running the NEMO3D application with different input problem sizes (emulating independent VIOLINs used by different users). Each VIOLIN starts executing the application at a different time. The shared infrastructure is comprised of two host domains. Domain 1 has six physical nodes while domain 2 has four physical nodes. The two domains are subsets of the two physical clusters in the nanoHUB. We do not yet have administrative privileges on any machines outside of Purdue's campus that can be used for these experiments, therefore we cannot experiment with truly wide-area infrastructures. However, the two domains that we are using are on separate subnets.

The experiment compares the execution time of NEMO3D within each VIOLIN with and without autonomic resource reallocation enabled. When reallocation is enabled, some VIOLINs will be migrated in accordance with the *adaptation manager's* heuristic in order to balance the load and improve the performance of applications.

Results. Figure 3 is a time-line showing where each VIOLIN environment is located at key time instances. Figure 4 shows recorded NEMO3D execution time of each VIOLIN with and without adaptation enabled.

Initially, for both runs, VIOLINs 1, 2, and 3 (referred to as V1, V2, and V3) are executing their applications and have been allocated significant portions of the host domains (referred to as D1 and D2). Each virtual machine is using nearly 100% of its allotted CPU.

V2 is executing a smaller problem size and is running alone in D2 so it finishes quickly. When V2's finishes, a load imbalance between the domains occurs. There are 10 virtual machines in D1 that expect more CPU allocation while there is no virtual machine in D2. The imbalance triggers the migration of V1 to the hosts of D2. This adaptation balances the load and allows the virtual machines of both V1 and V3 to be allocated the full resources of a single host.

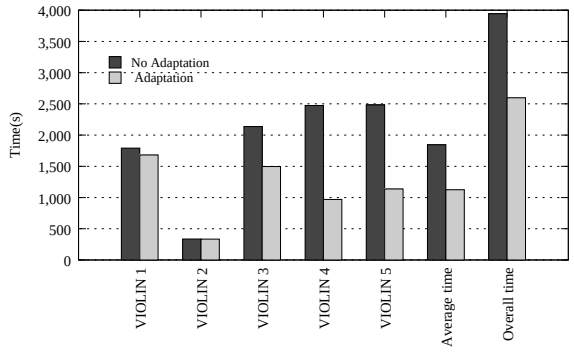

Figure 4. VIOLIN Adaptation Scenario 1: Execution time of applications running within VIOLIN environments with and without adaptation enabled.

It is important to note that although both remaining VIOLINs have increased CPU allocation, V1 temporally slows down during the migration. V3 will surely complete its application sooner, but it remains to be seen if the increased resource allocation to V1 can compensate for the cost of migration.

After some time, V4 and V5 start their applications and require significant resources (100% utilization). We assume that both of these environments are new and must be created to allow the non-adaptation case to have some balance in load. Without this allowance, V4 and V5 would have to remain where they were (potentially within D1 creating an even larger advantage for the adaptation case). In either case, the creation of V4 and V5 causes both domains to be overloaded, however, the load is balanced.

Next, V1 and V3 finish their applications. From Figure 4, we see that the migration of V1 allows V3 to finish 30% sooner than it would have otherwise, while V1 finishes in approximately the same amount of time due to the additional cost it pays to migrate. Once V1 and V3 finish, the remaining VIOLINs (V4 and V5) are already balanced in the adaptation case, while they are not in the non-adaptation case.

The chart in Figure 4 shows the application execution in each VIOLIN. For each VIOLIN, the execution time is reduced by enabling autonomic adaptation. The last two data points on the chart show the *average time* and *overall time* metrics of the system. The *average time* is the average execution time for all VIOLINs. In this example, adaptation saves on the average 39% of the application's execution time. The *overall time* is the duration between the execution of the first VIOLIN and the completion of the last VIOLIN. The *overall time* gives us a measure of the efficiency of resource usage. We see a 34% reduction in *overall time* with adaptation.

Discussion. Observe that during this experiment nearly all of the VIOLINs benefit from adaptation even though only one is migrated, suggesting that a small amount of adaptation can lead to a large increase in both application performance and resource utilization. In addition, heuristics that aim to balance load while minimizing the cost of migration are likely to achieve satisfactory performance without having to find the optimal allocation of resources to virtual machines.

4.3 VIOLIN Adaptation Scenario 2

Objective. Whereas the previous example shows the typical case where virtual environments are either being heavily used or completely idle, the next example shows how adaptation can benefit applications that go through periods of high and low use during a single execution. In this situation, we create a VIOLIN that initially uses a high amount of CPU then moves to a stage in its application that uses lower amounts of CPU.

Configuration. The configuration uses the same host infrastructure as the previous example. However, the VIOLINs and their applications have changed. There are now four VIOLINs, all of which execute the NEMO3D application except for V1. V1 executes the high demand NEMO3D followed by a less CPU intensive "dummy" application. V1 is simulating 100% utilization followed by a lower utilization that stabilizes within the desired utilization range after the appropriate reduction in CPU allocation.

Results. The time-line in Figure 5 and the chart in Figure 6 show the resulting execution time of the applications with and without adaptation enabled. Initially, the load is balanced between the four VIOLINs which are running on the two domains. After some time, V3 completes its application and no longer requires resources. Next V1 enters its second, less CPU intensive, stage of its execution. In the new stage, V1's utilization of resources drops well below desired range. Its drop in CPU allocation results in a load imbalance between the two domains, forcing the *adaptation manager* to migrate V2 to D1. The migration balances the load between domains but causes an imbalance between the hosts of D1. Since it is now possible for all six virtual machines from V1 to be supported by only two of the available hosts, they are migrated to the hosts left vacant by V2.

The results in Figure 6 show that V1 and V2 execute in approximately the same amount of time while V3 and

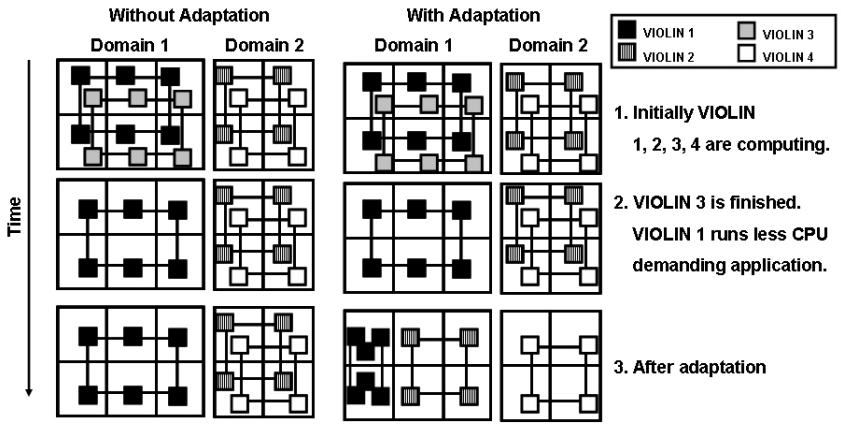

Figure 5. VIOLIN Adaptation Scenario 2.

V4 show significantly lower execution time. With autonomic reallocation enabled, the *average time* and *overall time* are decreased by 41% and 47%, respectively.

Discussion. From this experiment we see that it is possible to obtain further improvement of performance and efficiency by combining the fine-grained resource reallocation mechanisms with the coarse-grained migration mechanisms. The adaptation manager is able to identify virtual environments that experience a significant reduction in resource requirements. By scaling down the CPU share allocated to individual virtual machines of V1, it opens the possibility of migrating V2 thus improving the performance seen by all VIOLINs.

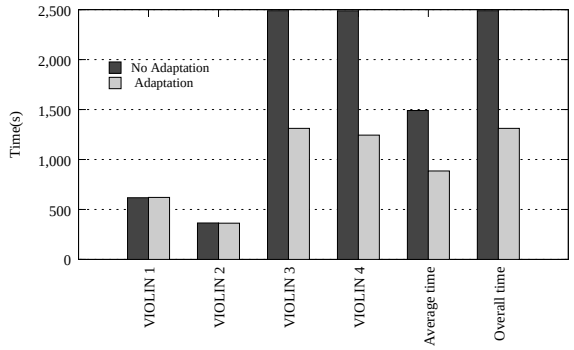

Figure 6. VIOLIN Adaptation Scenario 2: Execution time of applications running within VIOLIN environments with and without adaptation enabled.

4.4 VIOLIN Adaptation Scenario 3

Objective. The final adaptation scenario shows how the *adaptation manager* handles multi-stage applications with memory requirements that change during execution.

Configuration. In this example, the host infrastructure is limited to two domains each containing four hosts. Again, the VIOLINs execute NEMO3D which has two main stages, the first of which uses low amounts of memory while the second uses larger amounts of memory. During the execution, one of the VIOLINs (V3) doubles its memory usage from 160MB to 300MB when it transitions to the second phase of its execution.

Results. Figure 7 shows the time-line. Initially, V3 is in the first phase of its execution which uses a relatively low amount of memory (160MB). At this point, it is allocated a sufficient amount of memory (200MB), however its memory utilization is outside of the defined range. The *adaptation manager* determines that it needs to increase the memory allocation to the virtual machines in V3 in order to bring their utilization within the desired range. The *adaptation manager* sets the desired amount of memory to 400MB which cannot be satisfied by the hosts currently supporting the virtual machines. The *adaptation manager* is forced to move virtual machines and decides to migrate V1 to D2 allowing for the increase in V3's memory allocation. When V3's application reaches its second phase, its memory usage increases from 160MB to 300MB. Due to the adaptation, V3 has the memory it needs. Without adaptation the application would have crashed due to lack of available memory. In addition, when V3's application completes its second phase, it returns its excess memory, allowing V4 to be created.

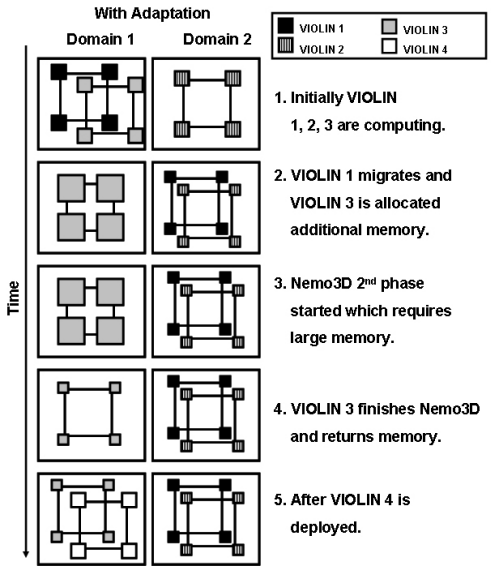

Figure 7. VIOLIN Adaptation Scenario 3.

Discussion. We recognize that any application can attempt to allocate an arbitrary amount of memory at any time and that we cannot predict this without knowledge of the particular application. For example, if V3's application needed to allocate 1GB of memory we would not have been able to support its request. The use of swap space would allow jobs to continue to run without enough allocated memory. A current limitation of our implementation is the lack of migration of virtual machine file systems, including swap partitions. Our future research will include file system migration which will allow swap partition migration, allowing us to monitor and adapt memory usage without any hard limits that may cause application failure.

5 Related Works

Currently, most techniques for federating and managing wide-area shared computation infrastructures apply meta-scheduling of Grid resources as in Globus [7], Condor [19], and In-VIGO [22]. All of these solutions provide access to large amounts of computational power without incurring the cost of full ownership. However, common to all of these systems is that arbitrary parallel/distributed applications cannot run unaltered through these systems and jobs run on nodes over which the users do not have administrative control.

In-VIGO is a distributed Grid environment supporting multiple applications that share resource pools. The In-VIGO resources are virtual machines. When a job is submitted, a virtual workspace is created for the job by assigning existing virtual machines to execute it. During the execution of the job, the virtual machines are assigned to the user who has access to his or her unique workspace through the NFS-based distributed virtual file system. Provided with In-VIGO is an automatic virtual machine creation service called VMPlant [13]. VMPlant is used to automatically create custom root file systems to be used in In-VIGO workspaces. In-VIGO is part of the nanoHUB deployment and can be made to use VIOLIN environments in the back-end.

Virtual networking is a fundamental component of VIOLIN. The available machine virtualization techniques do not supply advanced virtual networking facilities. UML, VMware, and Xen all provide networking services by giving the virtual machines real IP addresses from the host network. PlanetLab [2] uses a technique to share a single IP address among all virtual machines on a host by controlling access to the ports. These techniques allow virtual machines to connect to a network but do not create a virtual network. Among the network virtualization techniques are VIOLIN, VNET [16], and SoftUDC [11], all of which create virtual networks of virtual machines residing on distributed hosts. Of particular interest and relevance is VNET, which supports adaptation of network resources [17].

Cluster-on-Demand (COD) [3] allows dynamic sharing of resources between multiple clusters. COD reallocates resources by using remote-boot technologies to reinstall preconfigured disk images from the network. The disk image that is installed determines which cluster the nodes will belong to upon booting. In this way COD can redistribute the resources of a cluster among several logical clusters sharing those resources.

6 Conclusion

We have presented the design and implementation of adaptive VIOLIN virtual environments on top of a multi-domain shared infrastructure. Using our adaptation mechanisms and policies, virtual computation environments can move through the multi-domain shared infrastructure and adapt to the needs of their applications and availability of infrastructure resources. The *adaptation manager* acts on behalf of the users and infrastructure administrators to dynamically control the allocation of resources to the virtual environments. Our experiments with deployment of VIOLIN in the nanoHUB have shown significant improvement in application performance and resource utilization.

7 Acknowledgments

We would like to thank the anonymous reviewers for their constructive comments and suggestions. This work was supported in part by NSF Grants OCI-0438246, OCI-0504261, and CNS-0546173.

References

[1] 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 *ACM SOSP*, 2003.

[2] A. Bavier, 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 Network Services. In *USENIX NSDI*, 2004.

[3] J. Chase, D. Irwin, L. Grit, J. Moore, and S. Sprenkle. Dynamic Virtual Clusters in a Grid Site Manager. In *IEEE HPDC*, 2003.

[4] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of Virtual Machines. In *USENIX NSDI*, 2005.

[5] Jeff Dike. User-Mode Port of the Linux Kernel. In *Proceedings of the USENIX Annual Linux Showcases and Conference*, 2000.

[6] R. Figueiredo, P. Dinda, and J. Fortes. A Case for Grid Computing on Virtual Machines. In *IEEE ICDCS*, 2003.

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

[8] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[9] I. Foster and C. Kesselmann. Globus: A toolkit-based grid architecture. *The Grid: Blueprints for a New Computing Infrastructure*, pages 259–278, 1999.

[10] X. Jiang and D. Xu. VIOLIN: Virtual Internetworking on OverLay INfrastructure. Technical report, Purdue University, July 2003.

[11] M. Kallahalla, M. Uysal, R. Swaminathan, D. Lowell, M. Wray, T. Christian, N. Edwards, C. Dalton, and F. Gittler. SoftUDC: A Software-Based Data Center for Utility Computing. *IEEE Computer*, 37(11):38–46, 2004.

[12] G. Klimeck, F. Oyafuso, T. Boykin, R. Bowen, and P. von Allmen. Development of a Nanoelectronic 3-D (NEMO 3-D) Simulator for Multimillion Atom Simulations and Its Application to Alloyed Quantum Dots. *CMES 2002*, 3(5):601–642, 2002.

[13] I. Krsul, A. Ganguly, J. Zhang, J. Fortes, and R. Figueiredo. VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing. In *Supercomputing*, 2004.

[14] P. Ruth, X. Jiang, D. Xu, and S. Goasguen. Virtual Distributed Environments in a Shared Infrastructure. *IEEE Computer*, 38(5):63–69, May 2005.

[15] P. Ruth, P. McGachey, and D. Xu. VioCluster: Virtualization for Dynamic Computational Domains. In *IEEE CLUSTER*, 2005.

[16] A. Sundararaj and P. Dinda. Towards Virtual Networks for Virtual Machine Grid Computing. In *Virtual Machine Research and Technology Symposium*, pages 177–190, 2004.

[17] A. Sundararaj, A. Gupta, and P. Dinda. Increasing Application Performance In Virtual Environments Through Run-time Inference and Adaptation. In *IEEE HPDC*, 2005.

[18] G. Tesauroa, D. Chess, W. Walsh, R. Das, A. Segal, I. Whalley, J. Kephart, and S. White. A Multi-Agent Systems Approach to Autonomic Computing. In *AAMAS*, 2004.

[19] D. Thain, T. Tannenbaum, and M. Livny. Distributed Computing in Practice: The Condor Experience. *Concurrency and Computation: Practice and Experience*, 2004.

[20] VMware. http://www.vmware.com.

[21] S. White, J. Hanson, I. Whalley, D. Chess, and J. Kephart. An Architectural Approach to Autonomic Computing. In *IEEE ICAC*, 2004.

[22] J. Xu, S. Adabala, and J. Fortes. Towards Autonomic Virtual Applications in the In-VIGO System. In *IEEE ICAC*, 2005.