2006

# Enabling Autonomic Adaption of Virtual Computational Environments in a Shared Distributed Infrastructure

Paul Ruth

Junghwan Rhee

Dongyan Xu
*Purdue University*, dxu@cs.purdue.edu

Rick Kennell

Sebastien Goasguen

Report Number:
06-004

# ENABLING AUTONOMIC ADAPTION OF VIRTUAL COMPUTATIONAL ENVIRONMENTS IN A SHARED DISTRIBUTED INFRASTRUCTURE

**Paul Ruth**
**Junghwan Rhee**
**Dongyan Xu**
**Rick Kennell**
**Sebastien Goasguen**

**CSD TR #06-004**
**January 2006**

# Enabling Autonomic Adaptation of Virtual Computational Environments in a Shared Distributed Infrastructure

Paul Ruth, Junghwan Rhee, Dongyan Xu, Rick Kennell, Sebastien Goasguen
Department of Computer Science and ITaP
Purdue University
West Lafayette, IN 47907, USA
{*ruth, rhee, dxu*}*@cs.purdue.edu,* {*linux, sebgoa*}*@purdue.edu*

## 1 Abstract

A shared distributed infrastructure is formed by federating computation resources from multiple domains. Such a shared infrastructure provides aggregated computation resources to a large number 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 take 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 re-locate 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. We present the design, implementation, and evaluation of a middleware system that enables autonomic adaptation of virtual computation environments in a shared multi-domain infrastructure. A autonomic adaptive virtual computation environment, called a VIOLIN, is composed of a virtual network of virtual machines capable of *live migration across a multi-domain physical infrastructure.* To the best of our knowledge, this is the first demonstration of such capability using real-world parallel applications. Experimental results based on our real-world system deployment show improved performance of off-the-shelf scientific applications running inside autonomic adaptive VIOLINs.

## 2 Introduction

We have seen the emergence of shared distributed infrastructures that federate, allocate, and manage heterogeneous resources across multiple network domains, the most notable examples being PlanetLab [2] and the Grid [8, 9, 7]. 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, 21] has been increasingly adopted on top of such a shared physical infrastructures [6] achieving an elevated level of customizability, mutual isolation, and administrator privilege for users running their applications inside individual virtual machines.

Going beyond individual virtual machines, we, in our previous work, proposed techniques for the creation of virtual distributed computation environments [10, 15, 16] on top of a shared distributed infrastructure. A virtual computation environment, called a VIOLIN, is composed of virtual machines connected by a virtual network, providing 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 environments bearing the same look-and-feel of customized physical environments with administrative privilege (e.g., their own private cluster). Within the VIOLIN, the user is able to execute and interact with unmodified parallel/distributed application, with strong confinement of negative impacts by, possibly, untrusted applications.

The all-software virtualization of distributed computation environments brings the following unique opportunity to advance the possibilities enabled by autonomic systems [14, 22, 19]: it is possible to realize virtual computation environments as integrated, autonomic entities that dynamically adapt and re-locate themselves for better performance of the applications running inside. Note that such "on the fly" autonomic adaptation is not possible in a purely physical system. The autonomic adaptation of virtual computation environment is driven by two main factors: (1) dynamic, heterogeneous availability of infrastructure resources and (2) dynamic resource needs of the applications running inside the VIOLINs. Dynamic resource availability may cause a VIOLIN to re-locate its virtual machines to more resource-sufficient (e.g., CPU and memory) physical hosts when the current physical hosts experience increased workload, while dynamic applications may require different amounts of resources during its execution causing the VIOLIN dynamically adjusts its resource capacity to "catch up with"

the needs of the dynamic application. Furthermore, the autonomic adaptation (including re-location) of the virtual computation environment is *transparent* to the application and the user, giving the latter the illusion of a stable, well-provisioned, private, networked runtime environment.

To realize the vision of autonomic adaptive virtual computation environments in a multi-domain physical infrastructure we address the following challenges:

The first challenge is to provide the mechanisms for application-transparent virtual environment adaptation. In order to provide a consistent environment, adaptation must occur without effecting the application or the user. Work has be done to enable resource reallocation and migration within a local-area network [4] and many migration features are provided by the most current machine virtualization platforms. We still need to answer the question: how can we migrate virtual machines across a wide-area network without effecting the application? The solution must keep the virtual machine alive throughout the migration. Computation must continue and network connections must remain open. The necessary wide-area migration facility requires two feature not yet provided by current virtualization techniques. First, virtual machines need to retain the same IP address and remain accessible though the network despite physical routers not knowing where they were migrated. Second, wide-area migration cannot rely on NFS to maintain a consistent view of the large virtual machine images file. These files must be quickly be transfered across the relatively slow wide-area network. Current solutions, clearly, are not be adequate for wide-area use.

The second challenge is defining *allocation policies*: The ideal static allocation of shared resources considers the available resources and requested resources to finds the optimal allocation. However, autonomic environments must have the intelligence to scale resource allocations without user intervention. How do we know when a virtual machine needs more CPU? Which virtual machine should be migrated, if a host can no longer support the memory demands of its guests? If a virtual machine must be migrated where should it go? We must consider that 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 VIOLIN [15] enabled *autonomic virtual computation environments* that can be deployed over a wide-area shared infrastructures. These environments retain the customization and isolation properties of existing statically deployed VIOLINs, however, they have the added ability to autonomically adapt resource allocation driven by the dynamic needs of their applications without the application's knowledge. The environment, as well as the applications within the environment, will appear to be unchanged, except for its performance, even though it may have been migrated to distant host domain. In this way we can make efficient use of the available resources while giving the appearance of more powerful machines than actually exist. A autonomic adaptive virtual computation environment is composed of a virtual network of virtual machines capable of *live migration* across a multi-domain physical infrastructure.

We have built a prototype system using Xen [1] virtual machines and have deployed it over the NanoHub (www.nanohub.org) infrastructure. The performance evaluation 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 a autonomic adaptive virtual computation environment, using live application transparent migration with real-world parallel applications.

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

## 3 Autonomic Virtual Environments

We have designed VIOLIN *autonomic virtual environments* to address the dynamic needs of multi-domain shared infrastructures and their users. As in any multi-domain shared infrastructure, host domains participate by contributing varying numbers of heterogenous machines. However, unlike traditional shared infrastructures, the user's applications do not directly run on the host machines. Instead, each user is presented with an isolated autonomic virtual computation environment of virtual machines connected to an isolated virtual network overlay. From the user's point of view, the virtual computation environments are a static private subnet of machines dedicated to that user and are unaware of which hosts their virtual machines reside on. On the other hand, the infrastructure sees the environments as dynamic entities that can flow through the infrastructure being assigned as much or as little resources as needed.

Figure 1 shows an example shared infrastructure supporting multiple VIOLIN environments. The figure depicts two VIOLINs sharing a small wide-area infrastructure composed of machines from three independent domains. The users of VIOLINs A and B are unaware of each other or that they may be using the same physical hosts. The example depicts actions taken to counter the user of B initiating a CPU intensive executable on a virtual machine being shared with a virtual machine from VIOLIN A. If there were enough resources available, the virtual machine's local CPU allocation could be increased. However, in this case there is another virtual machine sharing the host and there isn't enough available CPU. One of the two machines on the host must be moved. In this case, VIOLIN A's virtual machine is migrated to a suitable host in another domain remedying the situation.

The key feature of VIOLIN *autonomic virtual environments* is dynamic live resource scaling and migration during application runtime. Over time, the properties of the virtual environments and the underlying infrastructure will change. Host machines may be added or removed. Virtual

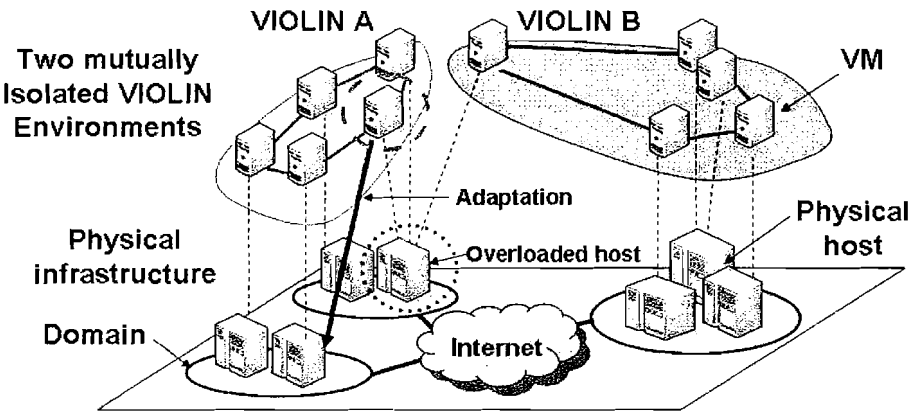


**Figure 1. VIOLIN environments sharing a multi-domain infrastructure. Virtual machines can migrate between domains to maintain sufficient resources for their changing applications.**

environments may be created, destroyed, or left idle; and the applications that they host may experience drastic changes in resource needs.

The components of the VIOLIN autonomic virtual computation environment system are:

- **Enabling Mechanisms**: The enabling virtualization-based mechanisms include the VIOLIN virtual environments as well as the *monitor daemon* running on the host infrastructure. The VIOLIN environments provide an interface to the user and applications, while the *monitor daemons* know the CPU power and memory available on each node and have the ability to query the local *virtual machine monitor* (VMM) for resource availability and utilization levels. In addition, the monitors can manipulate the allocation of resource to local guest machines.
- **Adaptation Manager**: The *adaptation manager* uses the *monitor daemons* to form a global system-view of all host resources available as well as the utilization level of any allocated resources. With this information the *adaptation manager* can dictate resource re-allocation including fine-grain per-node CPU and memory adjustments as well as coarse-grain migration of virtual machines or whole virtual environments without any user or administrator involvement.

The remainder of this section describes these components.

### 3.1 Enabling Mechanisms

The enabling mechanism for autonomic virtual computation domain is the daemon residing on each host that have the capabilities to monitor local resource availability and utilization, as well as, manipulate the portion of local resource allocated to each hosted virtual machine.

**Local Adaptation Mechanism**. The *monitor daemons* act as an intermediate through which the *adaptation manager* can control all virtual machines. VIOLIN autonomic environments use both memory ballooning and weighted CPU scheduling to achieve fine-grain control over per node resource multiplexing. While a virtual machine is running, the *adaptation manager*, through the *monitor daemon*, can modify the memory footprint and percentage of CPU allocated.

Both VMware [21] and Xen [1] allow for memory *ballooning* which allows for dynamic reallocation of memory to virtual machines. In effect, the VMM can change the amount of memory allocated to each virtual machine while the machine is running. Additionally, modern machine virtualization platforms allow for the weighted CPU scheduling. The use of these advanced schedulers allows for the *adaptation manager* assign arbitrary amounts of CPU power to each individual virtual machine.

**Wide-area Adaptation Mechanism** The key contribution of VIOLIN to autonomic environments is the ability to re-allocate resources to live virtual machines by migrating them across wide-area 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]. Migration is limited to a layer-2 network because IP packet routing is not designed to handled mobile IP addresses. VIOLIN [15] solves this problem by creating a virtual layer-2 network that tunnels network traffic end-to-end between remote virtual machines. The overlay appears to these machines to be an isolated physical Ethernet LAN though which migration is possible.

In our autonomic system, each virtual computation environment has its own VIOLIN network overlay. As the virtual machines flow through the infrastructure they will remain connected to their original virtual network. We are among the first to provide a system that allows for live wide-area

migration of virtual machines without the need to modify network addresses or use proxies.

## 3.2 Adaptation Manager

The second major component of dynamic VIOLIN environments is the *adaptation manager*. The *adaptation manager* is the intelligent agent, or "puppeteer", acting on behalf of the users and administrators, making autonomic reallocation decisions. It is appointed two tasks: to compile a global system-view of the available resources from the data collected by the resource monitoring daemons and to use this data to transparently adapt the assignment resources to virtual environments without the knowledge of the environment's application or users.

### 3.2.1 Infrastructure Resource Monitoring

The *adaptation manager* monitors the complete system through querying the monitor daemons on each host. Using the monitors it maintains knowledge of all available nodes in addition to the demands of applications running within the VIOLINs. Overtime both the resources available to 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 very high or low CPU, memory, or network usage. In order for the *adaptation manager* to successfully allocate dynamic resources to virtual machines it must monitor the availability and utilization of the allocated resources.

### 3.2.2 Resource Reallocation Mchanism

Although system monitoring is a responsibility of the adaptation manager, its key function is to decide how to allocate and re-allocate resources to best serve the VIOLINs. Once it has collected the data from the monitors and has created the global system-view, it knows the resource availability of each host, the current resource usage of each environment, and the current allocation of resources. With this information the *adaptation manager* locates environments with an over or under allocation of resources and can adapt the allocation to provide better performance or reduce the allocation to more efficiently use the infrastructure.

### 3.2.3 Resource Reallocation Policy

The *adaptation manager's* re-allocation policy is based on observed host resource availability and virtual machine resource utilization. It uses a heuristic that aims to dynamically balance load between all domains within the system, them between hosts within each domain. We do not attempt to find the ideal allocation of resources to virtual machines, but to incrementally increase the performance of the system while minimizing the number of virtual machine migrations and the resulting overhead.

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

It may be that there are not enough resources in the entire infrastructure to supply each virtual machine with its *desired resource level*. In this case, we would like to achieve a weighted balance of load on each domain and host (more powerful hosts/domains will take on more load). Conveniently, a weighted balance of load on an under-utilized system will assure that all (or most) virtual machines will have been allocated their *desired resource level*. With this in mind, our reallocation policy is designed to balance the load between domains and hosts.

The *desired resource level* assigned to each virtual machine is derived from the information the the *adaptation manager* obtains from the host-level monitors. For each virtual machine the *adaptation manager* knows the amount of CPU, in Floating Point Operations Per Second (FLOPS), and memory allocated, and the percentage of the allocation the virtual machine is utilizing. We define a utilization greater than 75% to be *high utilization* and below 25% to be *low utilization*. The *desired resource level* is defined to be double the current allocation for *high utilization*, half the current allocation for *low utilization*, equal to the current allocation otherwise.

If at anytime a virtual machine is under allocated (i.e. its *desired resource level* is greater than its allocated resources) the *adaptation manager* triggers the global reallocation algorithm.

Intuitively, the algorithm finds the average load on the whole infrastructure and attempts to migrate VIOLINs between domains until each domain has the load as the system average. Then, within each domain, virtual machines are migrated until each host has the domains average load. We define the *average system load* as the ratio of the total amount of *desired resources* for all virtual machine in the system to the total amount of resource provided by all hosts in the system. For each domain, we define the *average domain load* is the ratio of the total amount of *desired resources* for all virtual machine in the domain to the total amount of resources provided by all hosts in the domain. For each host the load is the ratio of *desired resources* to provided resources. To handle multiple types of resources that comprise these totals, the system declares a weight to be assigned to each type and the total is the weighted sum.

The algorithm is as follows:

1. Find the *average system load*
2. For each domain, find the *average domain load*. We want to reduce the load on domains whose average load is greater than that of the system by migrating of whole virtual environments to under-loaded domains.
3. Find inter-domain environment migration opportunities. Rank the domains by average domain load and find the VIOLIN from the most loaded domain that can

be migrated to the least domain, such that that both domains' average load becomes closer to the average system load.

4. Repeat steps 2 and 3 until step 3 produced no possible migrations. At this point, each domain has approximately the same domain load.

5. For each host, find the host load.

6. Find intra-domain virtual machine migration opportunities. Rank the hosts by average load and find the virtual machine with the most demand that can be migrated from the most loaded to the least loaded host, such that both host's loads become closer to the average domain load.

7. Repeat steps 5 and 6 until step 6 produces no possible migrations. At this point, each host within each domain has approximately the same host load.

# 4 Implementation

We have implemented a prototype *wide-area dynamic virtual environment* system and have deployed the system on the NanoHub's (www.nanohub.org) infrastructure. The NanoHub is an e-science infrastructure for running online and on-demand Nanotechnology applications, and is our "living lab". Part of the NanoHub allows students and researchers the ability to use computational Nanotechnology applications, including distributed and parallel applications, 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 of 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 applications.

## 4.1 Deployment Details

Toward a full deployment, we have created several prototype autonomic VIOLINs on the NanoHub's infrastructure.

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

**Environment Configuration.** Each environments is composed of several Xen virtual machines connected with a VIOLIN network overlay. Environments are composed of one virtual head node and several virtual compute nodes. The head node provides an access point 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 long as stay connected to the VIOLIN overlay.

User accounts for all machines 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 dynamic compute nodes, giving a consistent system view from all virtual machines regardless of the physical location of the virtual machine.

In order to migrate a virtual machine three things must be transfered to the new host: a snapshot of the root file system image, a snapshot of the current memory, and the thread of control. Xen's contribution to live migration is to very efficiently transfer the memory thread of control. It performs an iterative process that reduces the amount of time the virtual machine is unavailable to be almost unnoticeable.

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). Wide-area shared infrastructures are composed of independently administered domains which cannot safely share NFS servers. In order, to perform wide-area migrations, our prototype uses read-only root images that can be distributed without needing to be updated. We do this by putting all system file that need to be written to in *tmpfs* filesystems. Since, *tmpfs* file systems are resident in memory, Xen will migrate the files with the memory. Initially, we thought of this solution as a workaround to be fixed later, however, our experience is demonstrating that *tmpfs* can be a very good solution for many applications. In addition to the using *tmpfs* for system files, users home directories are NFS mounted through the virtual overlay to the NanoHub server and do not need to be explicitly transfered.

# 5 Experiments

In this section we present several experiments that show the feasibility of VIOLIN environments. First we measure the overhead of live migration of whole VIOLIN environments, then we measured increased performance due to autonomic adaptation of several examples of VIOLINs sharing a multi-domain infrastructure.

For all experiments we use the NanoHub VIOLIN deployment, an *adaptation manager* employing the algorithm described in section 3.2.3, and the NEMO3D [12] atomic particle simulation.

## 5.1 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 [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

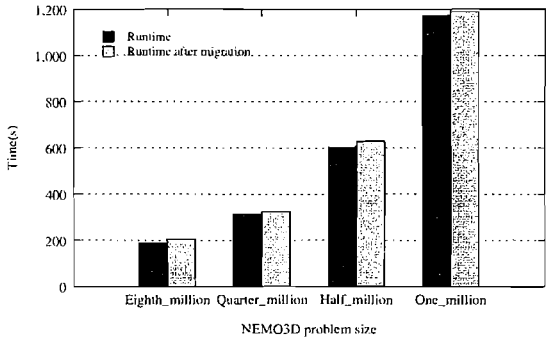


**Figure 2. Migration overhead caused by live migration of entire VIOLIN virtual environments that are actively executing the parallel application NEMO3D**

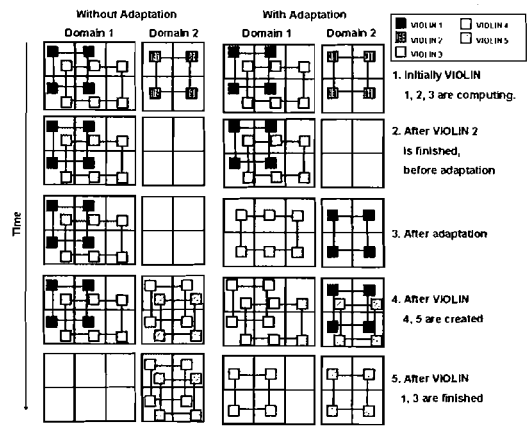


**Figure 3. Workload Adaptation Example 1: Time-line of VIOLIN's progress.**

overlay, and any lingering effects such as network slowdown caused by packet loss and the resulting TCP back-off.

**Configuration**. We used a VIOLIN composed of four virtual machines. We executed NEMO3D with several different problems sizes (1/8 Million Particles, 1/4 Million Particles, 1/2 Million Particles, 1 Million Particles). For each problem size we recorded the execution time with and without migrating the VIOLIN. During the no-migration runs, the application was 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 chose the migration to occur at the most resource intensive period of the application's execution. While the tests where occurring there was no background load any of the hosts involved, however the network is shared by many users and had some amount of unavoidable load. In addition, both CPU and memory are sufficiently provided to all virtual machines.

**Results**. Figure 2 shows the results. We found that, regardless of problem size, the runtime of the application was increased by approximately 20 seconds (ranging from 17-25) when the VIOLIN was migrated.

**Discussion**. One goal of adaptive VIOLIN environments is that there should be little or no effect on the applications due to adaptation. We observed approximately a 20 second penalty imposed on a four node VIOLIN migrating across a campus while running NEMO3D. A 20 second penalty would seem impossible considering Xen virtual machine migration requires the transfer of the entire memory footprint (approximately 800MB per virtual machine for the 1 Million particle NEMO3D). However, Xen's live migration facility hides the migration latency by continuing to run the virtual machine on the source host while the bulk of the memory is transfered. We didn't 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 on a LAN [4]. The significant effects on application performance are not the migration itself but the time to re-establish the VIOLIN network overlay and the additional time the application is running on the inadequate resources of the original hosts. This experiment shows that penalty for migrating a VIOLIN environment is relatively small and does not increase with increased virtual machine size.

## 5.2 Workload Adaptation Example 1

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

**Configuration** We created five VIOLIN environments, each of which is used to run the NEMO3D application. Each VIOLIN initiates its application at a different time with different input problem sizes (emulating independent VIOLINs used by different users). The shared infrastructure is comprised of two host domains. Domain 1 has 6 physical nodes while domain 2 has 4 physical nodes. The two domains are on separate sub-nets within Purdue's campus. 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 sub-nets confirming wide-area migration is possible.

The experiments compares the run-times of the NEMO3D applications within each VIOLIN with and without autonomic resource re-allocation enabled. With re-allocation enabled some virtual machines of the VIOLINs will be migrated in accordance with the *adaptation manager's* algorithm in order to balance the load and increase the performance of all applications.

**Results**. Figure 3 is a time-line showing where each virtual environment was located at key moments. Figure 4 shows recorded runtime comparisons with and without adap-

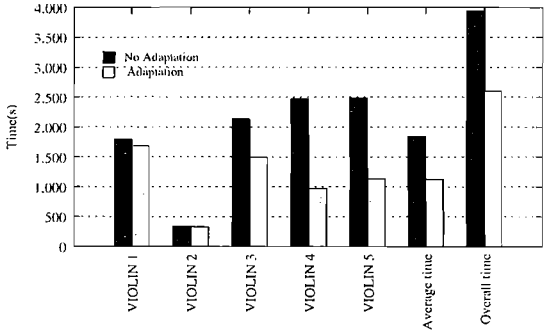


**Figure 4. Workload Adaptation Example 1: Runtime of applications running within VIOLIN environments with and without adaptation enabled.**

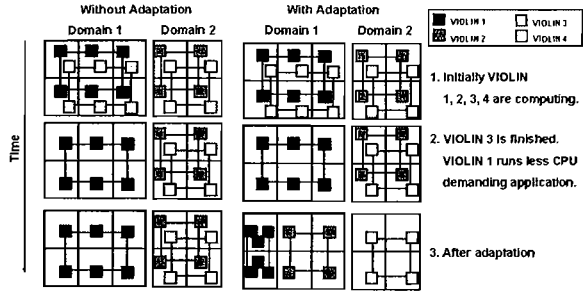


**Figure 5. Workload Example 2: Time-line of VIOLIN's progress.**

tation enabled.

Initially, for both runs, VIOLINs 1, 2, and 3 are executing their applications and have been allocated significant portions of the host domains. Each virtual machine is using nearly 100% of its alloted CPU. The *adaptation manager* sees the high CPU utilization and tries to allocated additional CPU for each virtual machine but the CPU load is balanced.

VIOLIN 2 is executing a smaller problem sized and is running alone on domain 2 so it finishes quickly. When VIOLIN 2's application finishes it remains on domain 2 but has nearly no requirements for resources. The *adaptation manager* reacts to the low (less that 25%) utilization and lowers VIOLIN 2's desired resource level. When VIOLIN 2's desired CPU power drops a load imbalance between the domains occurs. There are 10 virtual machines on domain 1 that desire increases CPU allocation and no virtual machines on domain 2 that need any CPU allocation. The imbalance, triggers the migration of VIOLIN 1 to the unallocated resources of domain 2. This adaptation balances the load and allows the virtual machines of both VIOLINs 1 and 2 to each be allocated the full resources of a single host. Although both VIOLIN 1 and 3 have been allocated additional CPU power they both remain at 100% CPU utilization but there are no resources for the *adaptation manager* to give.

It is important to note here that although both remaining VIOLINs have increased CPU power, VIOLIN 1 was temporally slowed during the migration. VIOLIN 3 with surely complete its application sooner, but it remains to be seen if the increased speed seen by VIOLIN 1 can compensate for the cost of migration.

After some time, VIOLINs 4 and 5 initiate their applications and require significant resources (100% utilization). We assume that both of these environments are new and must be created allowing the non-adaptation case to have some balance in load. Without this allowance, VIOLINs 4 and 5 would have to remain where they were (potentially within domain 1 creating an even larger advantage for the adaptation case). In either case, the creation of 4 and 5 causes both domains to be overloaded, however, the load is balanced.

Next, 1 and 3 finish their applications and no-longer require significant resources. From figure 4 we see that the migration of VIOLIN 1 allows VIOLIN 3 to finish 30% sooner than it would have otherwise, while 1 finishes in approximately the same amount of time due to the additional cost it paid to migrate. Once 1 and 3 finish, the remaining VIOLINs (4 and 5) are already balanced in a adaptation case, while the non-adaptation case they are not. Although the adaptation algorithm was lucky to create this state, its luck was not needed because an unbalanced state could have been corrected through migration unlike the non-adaptation case's current situation. Both cases continue to run and the adaptation case completes 4 and 5's applications much sooner.

The chart in figure 4 show the run-times for the applications in each VIOLIN. For each VIOLIN the runtime of the application is reduced by enabling autonomic adaptation. The last two data points on the chart show the *average time* and *overall time* performance of the system. The *average time* is the average run-time for all VIOLINs. This gives us a measure of the performance seen by each of the environments. In this example, adaptation saved on average 39% of execution time, correlating to a 39% average increase in performance seen by the environments. The *overall time* is the time from the execution of the first application until the completion last application (*overall time* is much less than *average time* because the applications are running in parallel). The *overall time* gives us a measure of the efficiency of resource use. 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 was migrated. It is important to realize that a small amount of adaptation can lead to large increases in both virtual environment performance and system efficiency. In addition, algorithms, such as ours, that aim to balance load while minimizing the cost of migration can have great effects on the performance of the system without needing to find and implement the ideal allocation of resources to virtual machines.

## 5.3 Workload Adaptation Example 2

**Objective**. Whereas the previous example shows the more typical case where virtual environments are being heavily used or are 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 high amounts of CPU then move to a stage in its application that uses lower but significant amount of resources.

**Configuration**. The configuration uses the same host infrastructure as the previous example, however, the VIOLINs and their applications have been changed. There are now 4 VIOLINs, all of which execute the NEMO3D application except for VIOLIN 1. Environment 1 executes the high demand NEMO3D followed by a less CPU application that searches the filesystem. VIOLIN 1 simulates a 100% utilization followed a lower utilization that stabilizes at or around 50% after the appropriate reduction in CPU allocation.

**Results**. The time-line in figure 5 and the chart in figure 6 show the resulting run times of the experiment applications with and without adaptation enabled. Initially, the load is balanced between the 4 VIOLINs which are running on the 2 domains. After some time, VIOLIN 3 completes its application and no longer requires significant resources (its CPU allocation is slowly reduced to near zero). Next VIOLIN 1 enters its second, less CPU intensive, stage of its execution. In the new stage VIOLIN 1's demand for resources drops well below 25% of its allocation. Its drop in CPU allocation results in load imbalance between the 2 domains forcing the *adaptation manager* to migrate VIOLIN 3 to domain 1. The migration balances the load between domains but causes an imbalance between the hosts of domain 1. Since it is now possible for all 6 virtual machines from VIOLIN 1 to be supported by only 2 of the available hosts, they are migrated to the hosts that are not supporting VIOLIN 2.

The results in figure 6 show that environments 1 and 2 execute in approximately the same amount of time while 3 and 4 show significantly decreased runtime. With autonomic re-allocation enabled, the *average time* and *overall time* show decreases of 41% and 47% respectively.

**Discussion**. From this experiment we see that it is possible to obtain even more performance and efficiency by combining the fine-grain resource allocating mechanisms of machine virtualization with the large-grain wide-area migration mechanism. Here the algorithm was able to identify a virtual environment that experienced a significant reduction in resource requirements. By controlling the CPU power allocated to individual virtual machines of VIOLIN 1, it was able to open the possibility of migrating VIOLIN 2 increasing the performance seen by all environments.

## 5.4 Memory Driven Adaptation

**Objective**. The final adaptation example shows how the *adaptation manager* handles multi-stage applications

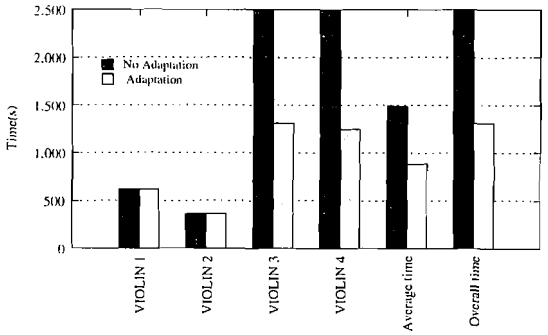


**Figure 6. Workload Adaptation Example 2: Runtime of applications running within VIOLIN environments with and without adaptation enabled.**

that have dramatically different needs for memory during different stages of there execution.

**Configuration**. In this example, the host infrastructure is limited to 2 domains each of which contain 4 hosts. We, again, use NEMO3D which has two main stages, the first of which uses very little memory and the second which uses a large amount. During the run one of the environments (VIOLIN 3) doubles its memory usage from 200MB to 400MB when it enters the second phase of its execution.

**Results**. Figure 7 shows the time-line. Initially, VIOLIN 3 is in the first phase of its application which uses a relatively low amount of memory (approximately 160MB). At this point, it has been allocated enough memory (200MB). The *adaptation manager* sees that VIOLIN 3 is using more than 75% of it allocated memory and determines that it should be increased to 400MB. VIOLIN 3's desire for increased memory causes an resource allocation to be imbalance forcing the *adaptation manager* to migrate VIOLIN 1 to domain 2. This migration allows for the necessary increase in VIOLIN 3's memory allocation. When VIOLIN 3's application reaches its second phase its memory usage increases from 160MB to 300MB. It has the memory it needs because adaptation was enabled, without adaptation enabled the application would have crashed due to lack of available memory. In addition, when VIOLIN 3's application completes its second phase it can then return its excess memory allowing a 4th environment to be created.

**Discussion**. This example shows how dynamically balancing load between domains can allow for applications to allocate memory and run where it is not possible without adaptation. 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 VIOLIN 3's application had gone from 160MB allocated to 1GB we would not have been able to support its request. The use of virtual memory and swap partitions would allow any job to continue to run (although much slower) without enough memory. A current limitation of our implementation is the lack of migration

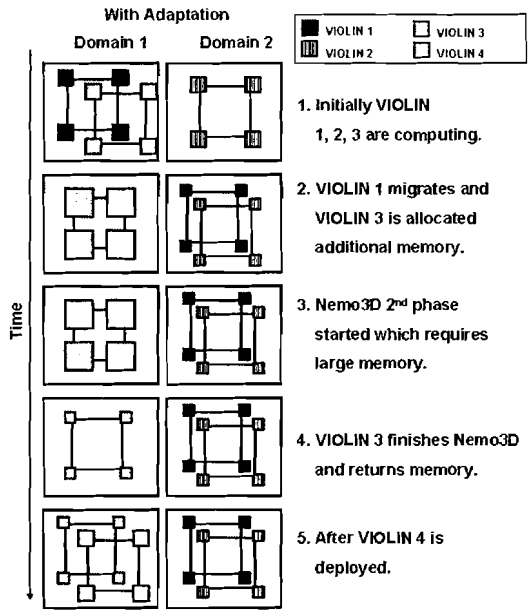


**Figure 7. Memory Example: Time-line of VIOLIN's progress.**

of updated virtual machine file systems, including swap partitions. Our future research will include file system migration which will allow swap partition migration and virtual memory use allowing us to monitor and adapt memory usage without any hard limits that can cause application failure.

## 6 Related Works

Currently, most techniques for federating and managing wide-area shared computation infrastructures apply meta-scheduling of dedicated Grid resources like Globus [8], Condor [20], and In-Vigo [23]. All of these solutions provide access to seemingly endless amounts of computational power without incurring the full cost of ownership. However, common to all of these systems is that arbitrary jobs cannot be run unaltered through these systems, jobs are run on nodes over which the job owner has no control, and allocation of resources cannot adapt to dynamic changes in application needs.

In-VIGO is a distributed Grid environment supporting multiple applications which 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 process it. During the execution of the job the virtual machines are owned by the user and the user has access to his or her unique workspace image through the NFS-based distributed virtual file system. Provided with In-VIGO is an automatic virtual machine creation project called VMPlants [13]. VMPlants 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 as a back-end.

Virtual networking is a fundamental part of our work. 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 a real IP address 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 [18], and SoftUDC [11] all of which create virtual network overlays of virtual machines residing on distributed hosts. The creators of VNET are currently working on dynamic 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.

## 7 Conclusion

We have presented the design and implementation of VIOLIN autonomic virtual computation environments for multi-domain shared infrastructures. Using VIOLINs, independently administered virtual computation domains can flow through the massive amount of computation resources available through multi-domain shared infrastructures adapting to the the needs of their applications. We have shown the design and implementation of the feature of VIOLIN environments that allows for wide-area migration of live virtual machines and the *adaptation manager* that acts on behalf of the users and administrators to dynamically control the allocation of all resources in the shared infrastructure. Our experiments with our NanoHub deployment of virtual computation environments has shown significant performance and efficiency increases. With continued advancement of machine and network virtualization, as well and resource allocation policies, VIOLIN virtual computation environments will continue to increase the potential of multi-domain shared infrastructures.

## References

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew 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.

[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, 2004.

[3] Jeffrey S. Chase, David E. Irwin, Laura E. Grit, Justin D. Moore, and Sara E. Sprenkle. Dynamic virtual clusters in a grid site manager. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, page 90, Washington, DC, USA, 2003. IEEE Computer Society.

[4] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of 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, 2003.

[7] I Foster, C Kesselman, J Nick, and S Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. In *Open Grid Service Infrastructure WG*. Global Grid Forum, 22 June 2002.

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

[9] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[10] Xuxian Jiang and Dongyan Xu. Violin: Virtual internetworking on overlay infrastructure. Technical report, Purdue University, 2003.

[11] Mahesh Kallahalla, Mustafa Uysal, Ram Swaminathan, David E. Lowell, Mike Wray, Tom Christian, Nigel Edwards, Chris I. Dalton, and Frederic Gittler. Softudc: A software-based data center for utility computing. *IEEE Computer*, 37(11):38–46, 2004.

[12] Gerhard Klimeck, Fabiano Oyafuso, Timothy B. Boykin, R. Chris Bowen, and Paul von Allmen. Development of a nanoelectronic 3-d (nemo 3-d) simulator for multimillion atom simulations and its application to alloyed quantum dots. *Computer Modeling in Engineering and Science (CMES)*, 3(5):601–642, 2002.

[13] Ivan Krsul, Arijit Ganguly, Jian Zhang, Jose A. B. Fortes, and Renato J. Figueiredo. Vmplants: Providing and managing virtual machine execution environments for grid computing. In *SC '04: Proceedings of the Proceedings of the ACM/IEEE SC2004 Conference (SC'04)*, page 7, Washington, DC, USA, 2004. IEEE Computer Society.

[14] H. Liu and M. Parashar. Enabling self-management of component based high-performance scientific applications. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC 2005)*.

[15] Paul Ruth, Xuxian Jiang, Dongyan Xu, and Sebastien Goasguen. Virtual distributed environments in a shared infrastructure. *IEEE Computer*, 38(5):63–69, May 2005.

[16] Paul Ruth, Phil McGachey, and Dongyan Xu. Viocluster: Virtualization for dynamic computational domains. In *CLUSTER 2005*, September 2005.

[17] A. Sundararaj, A. Gupta, and P. Dinda. Increasing application performance in virtual environments through run-time inference and adaptation. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC 2005)*, 2005.

[18] Ananth I. Sundararaj and Peter A. Dinda. Towards virtual networks for virtual machine grid computing. In *Virtual Machine Research and Technology Symposium*, pages 177–190, 2004.

[19] Gerald Tesauroa, David M. Chess, William E. Walsh, Rajarshi Das, Alla Segal, Ian Whalley, Jeffrey O. Kephart, and Steve R. White. A multi-agent systems approach to autonomic computing. In *Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1 (AAMAS'04)*.

[20] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: The condor experience. *Concurrency and Computation: Practice and Experience*, 2004.

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

[22] Steve R. White, James E. Hanson, Ian Whalley, David M. Chess, and Jeffrey O. Kephart. An architectural approach to autonomic computing. In *ICAC 2004*.

[23] Jing Xu, Sumalatha Adabala, and Jose A. B. Fortes. Towards autonomic virtual applications in the in-vigo system. In *Proceedings of the 2nd IEEE International Conference on Autonomic Computing (ICAC-05)*, pages 15–26, June 2005.