VioCluster: Virtualization for Dynamic Computational Domains

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

Abstract

A large organization, such as a university, commonly supplies computational power through multiple independently administered computational domains (e.g. clusters). Each computational domain faces the conflict between dynamic workload and static capacity. This is clearly inefficient at times when some clusters have idle nodes while others experience excessive workload. An opportunity arises to resolve this conflict by dynamically adapting the capacity of clusters by borrowing idle machines of peer domains. In this paper, we present the design, implementation, and evaluation of VioCluster, a virtualization based computational resource sharing platform. Through machine and network virtualization, VioCluster enables virtual computational domains *that safely "trade" machines between them without infringing on the autonomy of either domain. Our performance evaluation results show that dynamic machine trading between virtual domains increases their resource utilization and decreases their job wait times.*

1 Introduction

To meet the varied computational needs of a large organization it is often necessary to maintain multiple separate computational domains. These domains are administered independently, and will have software, hardware and network environments customized to best serve their organizational unit. The workloads assigned to these clusters will also vary; while one cluster is experiencing a spike in workload, another may be sitting idle. Clearly this is wasteful of computational resources.

This wastage could be decreased were the organization able to temporarily transfer resources from an underutilized domain to a busy one. Once the period of peak activity has ended, these nodes could be returned to the original domain. By borrowing resources during non-overlapping periods of heavy usage, the throughput of each cluster in an organization could be improved.

Realizing this goal, however, is not a simple task. Each domain will be configured according to the requirements of its owners. As a result nodes from different domains may not be able to inter-operate. Machines under different domains may have different software packages or user permissions. Worse, one domain may have access to hardware unavailable on another, or be on a private subnet to which other machines do not have access.

Further, organizational units may be unwilling to allow potentially unsafe code to run on their machines, particularly under a privileged account. By lending machines to another cluster, the safety and isolation of its own jobs may be threatened. However, without root access, it may be impossible for the borrowing cluster's jobs to run.

In this paper, we present VioCluster, a novel architecture which allows dynamic machine trading while avoiding these problems. We introduce the concept of *virtual computation domains* (or "*virtual domains*" for short) which allow a cluster to dynamically grow and shrink based on resource demand. Under this system, the administrative privileges of both the borrowing and lending clusters are maintained: cluster administrators are able to configure borrowed machines as required, while not granting root privileges to others making use of their nodes.

A VioCluster uses both machine and network virtualization techniques to logically move machines between virtual domains. Borrowed nodes in a VioCluster take the form of a virtual machine (VM) running on top of a host machine located in another *physical domain*. This VM process runs as an unprivileged user, fulfilling the requirement that administrative access remains exclusively with the node owner. However the configuration of the VM is determined by the administrator of the borrowing domain, allowing the ability to install software packages or hardware as required.

To users and applications, the process of borrowing nodes is transparent. A VM running as part of a VioCluster is practically indistinguishable from a physical machine running inside the same domain.

The VioCluster system offers several contributions:

- Dynamic machine trading between mutually isolated virtual domains. VioCluster creates software-based network components which seamlessly connect physical and virtual machines to create isolated virtual domains. Machines can be traded dynamically through the on-demand creation, deletion, and configuration of VMs and network components.

- Dynamic negotiation of machine trades. Each virtual domain includes a *machine broker* which interacts with other domains. Requests and offers are made through these brokers based on workload and configurable lending and borrowing policies.

We have built a prototype of the VioCluster system, and have demonstrated its effectiveness using two independent Portable Batch System (PBS) [11] based job-execution clusters. Our performance evaluation results show benefits to both clusters by increasing their resource utilization and decreasing their job execution times.

The remainder of the paper is organized as follows: Section 2 describes the design of VioCluster, Section 3 presents key implementation details, Section 4 describes the experiments and presents performance results, and Section 5 compares VioCluster with related work. Section 6 presents the paper's conclusions.

2 Design

There are two key components in the VioCluster system: the ability to create dynamic virtual domains and the mechanism by which trades are negotiated. This section describes the structure of these components, and the manner in which they interact.

A summary of the terminology used within VioCluster is as follows:

- **Physical domain**: An autonomous set of networked computers managed as a unit. Physical domains have a single administrator, and support a user-base performing specific computational activities. For example, a physical domain belonging to a biology department may be optimally configured for cellular simulations, while a physical domain belonging to a network research group may be designed for shorter network-intensive experiments.

- **Virtual domain**: An autonomous set composed of virtual and physical machines managed as a unit. Machines in a virtual domain are connected through a virtual private network, to which both virtual and physical machines have access. Virtual domains are able to grow and shrink on demand, and to the administrator appear to be identical to physical domains. A one-to-one mapping exists between physical and virtual domains; every virtual domain is hosted upon a physical domain.

- **Machine broker**: A software agent that represents a virtual domain when negotiating trade agreements with other virtual domains. A machine broker consists of a *borrowing policy* which determines under which circumstances it will attempt to obtain more machines, and a *lending policy* which governs when it is willing to let another virtual domain make use of machines within its physical domain. Both policies are defined by the domain's administrator.

Figure 1 shows an example VioCluster consisting of two physical domains, A and B. There are virtual domains associated with each physical domain. Both physical domains consist of 36 machines, each of which initially belongs to its respective virtual domain. These clusters could be imagined to belong to two university departments, or to two divisions within a company.

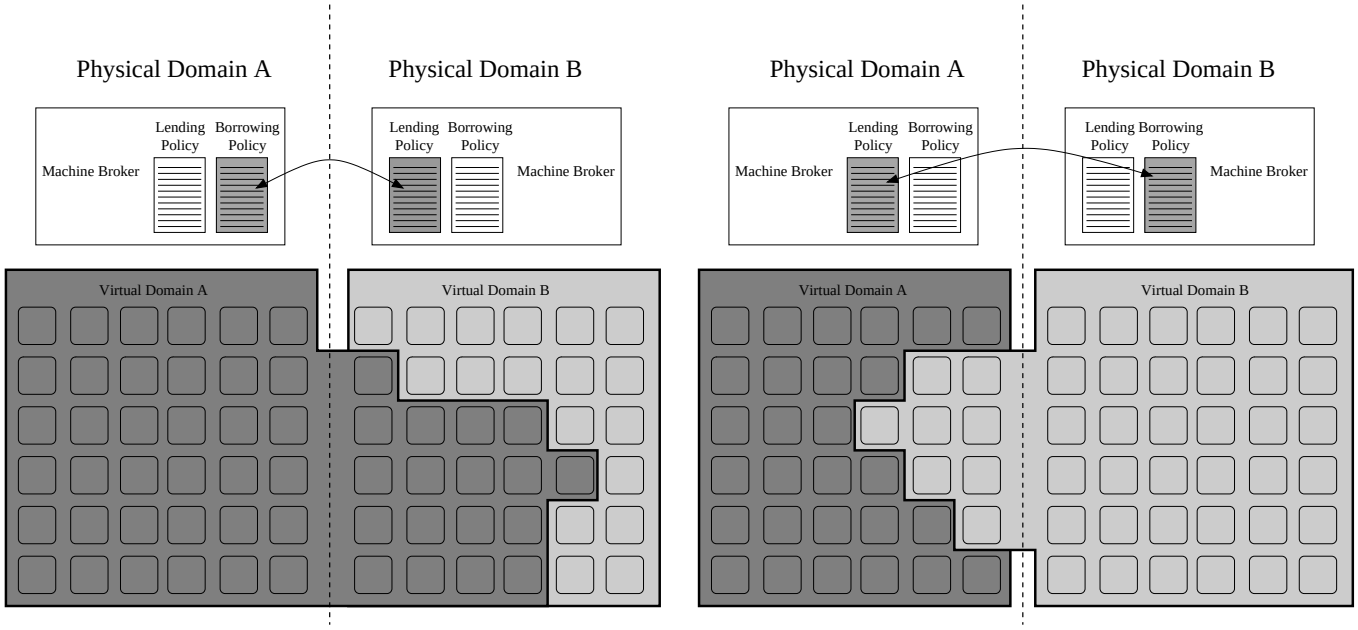
Over time, the workload on each domain varies as jobs are submitted by users. In Figure 1(a), virtual domain A is experiencing a period of heavy demand, while virtual domain B is under-utilized. After negotiations between the borrowing and lending policies of the respective brokers, virtual domain A is able to temporarily borrow half of virtual domain B's nodes. In Figure 1(b), the workload patterns are reversed, and virtual domain B is able to use nodes located in physical domain A.

When a machine belonging to physical domain B is borrowed by virtual domain A, it is used to run a virtual machine. This VM is owned by the administrator of virtual domain A, and will match the configuration of the machines in physical domain A. Virtual network connections will be made, connecting the new VM to the nodes of virtual domain A.

The trading process is authorized according to the borrowing and lending policies within the machine brokers of each domain. These policies are defined by the domain administrator, and allow complete control over the access to a domain's resources. Without an agreement between the brokers, the trade cannot occur.

2.1 Dynamic Virtual Domains

The use of virtualization is key within VioCluster. Through the use of virtual machines, many of the configuration and access problems inherent in machine trading can be avoided. Additionally, virtual networking allows physical and virtual machines to communicate transparently, making



(a) Virtual domain A borrowing nodes from virtual domain B in accordance with A's borrowing policy and B's lending policy

(b) Virtual domain B borrowing nodes from virtual domain A in accordance with A's lending policy and B's borrowing policy

Figure 1. Two virtual domain trading resources. Virtual domains fluidly trade machines in reaction to changes in workload.

network administration no more difficult than for a single physical cluster.

2.1.1 Machine Virtualization

When transferring physical machines between domains without the aid of virtualization, many problems may be encountered. The set of user and group accounts on clusters may be different, leading to access problems when running jobs. Necessary packages and services may not be installed, and superuser permissions may be required to customize a machine's configuration. Additionally, once a borrowed machine is no longer necessary, it must be restored to its original state before it can be used again in its original context.

VioCluster virtual domains bypass all these problems through the use of virtualization. When a physical node is lent to another virtual cluster, all that is required is a VM process run by an unprivileged user. The VM is created using a *disk image* supplied by the borrowing cluster. The user accounts, services and software services on this image can be configured identically to those on the virtual domain's physical machines. And when the machine is ready to be returned to its original cluster, all that is required to restore its state is the termination of the VM process.

2.1.2 Network Virtualization

Networking between nodes in a virtual domain is made possible by an enhanced version of the VIOLIN [7, 14] network overlay. Traditional VIOLIN creates a virtual layer-2 network overlay that tunnels network traffic end-to-end between remote virtual machines. The overlay appears to these machines to be an isolated physical Ethernet LAN. For VioCluster, we developed *Hybrid VIOLIN*, which adds the ability for physical machines to connect to this private network. As a result, virtual and physical machines are able to exchange network data transparently.

The underlying mechanisms of the Hybrid VIOLIN network is shown in Figure 2. Virtual and physical machines connect to the network through a *distributed virtual switch*. Network traffic is sent to a local *virtual switch daemon* by both virtual and physical machines, and is then forwarded to corresponding processes on other nodes of the virtual domain.

Figure 2 shows a physical and virtual machine interacting over a Hybrid VIOLIN network. Physical host A connects to a virtual NIC running in kernel space. This NIC,

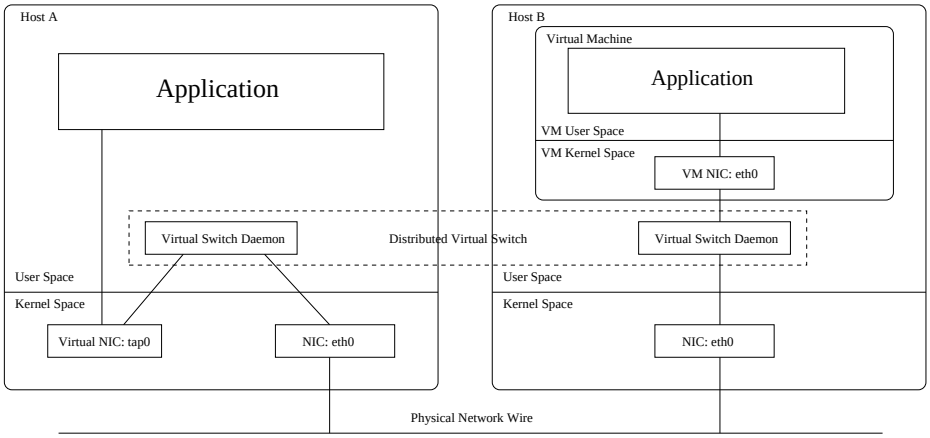


Figure 2. Hybrid VIOLIN

called a TUN/TAP device, appears to applications running on the physical host as an additional network interface. Data sent via standard system calls to the TUN/TAP device is then forwarded through the virtual switch daemon running in the user space of the host. The data is then sent out onto the virtual network.

The virtual machine running on host B, follows a similar connection procedure. In this case, the virtual NIC is resident in the kernel space of the virtual machine. Data is then forwarded through a virtual switch daemon in the host machine's user space, and is transmitted to other machines on the virtual network in the same way as from host A.

The virtual switch daemon receives Ethernet frames generated by the virtual network interfaces. It encapsulates these frames inside a UDP packet, and forwards them across the physical network to the virtual switch daemon on the destination physical node. At the destination, the frames are un-encapsulated and forwarded to the proper virtual interface. Throughout the process, virtual network traffic is never placed on the physical network, maintaining strict isolation between domains.

The effect of the Hybrid VIOLIN network overlay is that each virtual domain has a uniform and private IP address space. VMs running on borrowed nodes can be assigned IP addresses in the same range as the physical machines belonging to the domain. Maintaining this illusion simplifies the administration process and minimizes IP accounting. Additionally, because Hybrid VIOLIN virtualizes addresses at layer-2, arbitrary IP address spaces can be designated to virtual domains without the threat of conflicts with the host network.

In effect, Hybrid VIOLIN creates a private virtual LAN. Machines participating in a virtual domain have the same access to domain services as nodes in a physical domain. For example, all nodes on a virtual domain may have equal access to NFS-mounted directories.

The machine and network virtualization techniques used in VioCluster allows the isolation and safety required to make machine trading a viable proposition for large organizations. Fully customizable machine configurations running inside isolated VMs and private networks allow virtual domains that are borrowing nodes the flexibility to perform their work and ensures that physical domains that are lending nodes are not forced to compromise their security.

2.2 Machine Brokering

The second major component of the VioCluster system is the mechanism by which machine exchanges are negotiated. Each domain has a software agent called a *machine broker* which has the responsibility of determining whether trades should occur. The VioCluster system supplies the means by which physical domain administrators may define lending and borrowing policies. Wise decisions made in these policies can lead to large benefits in overall system throughput, while poor choices can degrade performance.

The policies used by a machine broker must be designed with several factors in mind. Observations of the current and past workload levels in the domain may allow predictions of future demand. For example, a cluster may see predictable spikes in usage during business hours, with less demand at night. In that case, a sensible set of policies might borrow during the day and lend at night. Additionally, knowledge of the applications run on a domain may influence policy decisions. For example, if a cluster is used primarily for short, network-intensive experiments, borrowing nodes may lead to unacceptable communication overheads.

What follows are several attributes that must be specified in the policies for negotiating machine trades:

- **Reclamation**: A policy must determine when a machine will be returned to its home domain. One recla-

mation policy may be to lend a machine for a specified lease period. Once this time is over, the node must be returned. Alternatively, a lending domain may wait until a remote job has completed, or reserve the right to reclaim machines gradually as its demand increases.

Another possible policy would be to lend a machine for an unlimited period, on the understanding that it will be returned when required by the owner. While this offers flexibility to the lending domain, it requires the borrower to be able to recover from the sudden loss of a machine. If the borrowing virtual domain cannot handle this situation, its borrowing policy should forbid such trades.

- **Machine properties**: The characteristics of the machine to be borrowed will have an impact on the policies of the domains involved in a trade. The machine broker must ensure that any machine lent to another virtual cluster has the processing power, memory and disk space required to run a virtual machine. Since the granularity at which resources are assigned is the machine, some properties, such as CPU power and memory, will remain constant. However others, such as available disk space, may change and must be monitored by the machine broker. Machines not capable of running a virtual machine suitable for the tasks required may be rejected by the borrowing machine broker.

- **Location**: For some applications, particularly those with high levels of communication between nodes, the physical location of machines may affect performance. Bandwidth and latency within a virtual domain may be affected by the location of communicating nodes. A borrowing policy should be aware of the communication requirements of its applications. If little communication is required, it may be acceptable to borrow nodes with high-latency or nodes with low-bandwidth connections to the remainder of the machines. Alternately, if the applications run on a virtual domain tend to be tightly-coupled with high levels of network traffic, it may be best to wait until nodes can be co-located on a single physical domain.

3 Implementation

We have implemented a prototype VioCluster system that uses a domain virtualization mechanism based on User Mode Linux (UML) [18] and Hybrid VIOLIN, which is governed by simple but effective machine brokering policies. It is worth mentioning that at the time of this writing we have implemented a Xen [2] version of VIOLIN. Our initial experience with VIOLIN and Xen indicate greatly improved performance over UML.

As an example application scenario, our prototype configures virtual domains as clusters managed by a PBS job scheduler. Within each virtual domain one physical machine is designated the PBS master node and the remainder of the machines are configured as PBS compute nodes. As workload changes, machines are added and removed from the virtual domain and the PBS master is re-configured to allocate jobs to all machines in the virtual domain. It is important to note that batch scheduling is *not* the focus of our work and PBS is only used as a sample application.

3.1 Domain Virtualization Implementation

User Mode Linux and Hybrid VIOLIN are well suited to be VioCluster's machine and network virtualization techniques because of their user level execution needed for isolation. In addition, Hybrid VIOLIN's networking abilities that support both virtual and physical machines fit VioCluster's unique needs.

Virtual Network Configuration. In our prototype, we define a virtual domain by a Hybrid VIOLIN network. The machines connected to each virtual domain's virtual network are a mixture of the real machines of its physical domain and the VMs created on nodes borrowed from other physical domains.

Each physical machine has two NICs: one connected to the physical LAN (*eth0*), and the other to the Hybrid VIOLIN network (*tap0*). Remote domains access the host via a private non-routable IP address associated with *tap0*. Depending on the configuration of the virtual NIC, VMs can have Internet routable IP addresses or private non-routable IP addresses through which they can access the Internet through a NAT router. The routing tables of the physical machine must be aware of the virtual network configuration, enabling traffic destined for the physical machine to be sent to *eth0* while virtual machine traffic is forwarded to *tap0*. For this reason, administrative privileges are needed on the physical domain to create and manage the TUN/TAP device.

Virtual Machine Configuration. VioCluster dynamically creates and destroys UML virtual machines at the request of the machine brokers. The VMs are configured by the borrowing domain to fulfill the needs of its applications. In the prototype system, all virtual domains use a modified Fedora Core I root file system. This includes all libraries, packages and applications (such as MPI and PBS) required to function like the physical machines.

Since the VMs are nearly-identical PBS compute nodes, small Copy-On-Write (COW) files can be used to capture the differences in their disk images. These COW files share a common root image and store only the minor differences between images, such as network settings. Upon creation, a COW system image represents an individual VM and can

be transfered to any host to be instantiated.

The prototype system uses a pool of pre-built COW file systems. However, it would also be possible to create them on-demand.

Virtual Machine Instantiation. The creation of a UML virtual machine requires only user-level access to the host machine. The root file system must be transferred, the virtual switch daemon started, and the VM booted. To this end, each physical domain maintains a user account for each peer virtual domain that may borrow machines.

The size of the root filesystem used in the prototype is approximately 300MB when compressed. Transferring this across the network whenever a VM is required would be prohibitive, so the base system is copied to each potential host before any VM is created. As a result, only the small (approximately 200KB) COW file must be transferred on demand.

Once the root image is transferred, the virtual switch daemon is started via an *ssh* connection. The daemon contacts the peer daemons on the other hosts, and joins the distributed switch. Finally, the VM is booted via *ssh*, and connects through the distributed virtual switch.

Virtual Machine Removal. Before a VM can be removed from a virtual domain, it must be halted. There are two ways this can be achieved: by killing the VM process or by using the *shutdown* command inside the virtual machine. Killing the process is faster, but results in a corrupted COW file. However, the common root disk image is read-only, so it is unaffected.

The state maintained on the VMs COW files is not important, since the VMs are used as PBS compute nodes. As such, when we need to create a new VM, we can simply copy the original COW file from the virtual domain administrator. This means that it is not necessary to properly shut down VMs, making the halting process faster.

3.2 Machine Brokering Implementation

Virtual domains share resources according to *contracts* agreed upon by their machine brokers. Trades are permitted or denied in accordance with the brokers policies, based on virtual domain demand levels. Virtual domains experiencing heavy workloads propose trade offers, while those with spare capacity advertise the capabilities of the machines they have available. Only if the trade is acceptable to the policies of both brokers will the proposal be accepted.

Our implementation of VioCluster uses PBS to schedule jobs. It should be noted that our intent is not to study job scheduling; we use PBS simply as a demonstration of VioCluster's ability to operate in a dynamic environment. Applications running on virtual domains must be able to handle changes in available machines. PBS, and most job schedulers, can adapt to these changes and are excellent ways to use the dynamic resources of virtual domains. When a machine is added or removed from the virtual domain the PBS master daemon is re-configured to reflect the change. One benefit of using a job scheduler is its inherent resilience to node failure. Nodes can be preempted at any time without effecting the correctness of the applications. Jobs will be re-run by the scheduler with the only effect being on performance. To reduce the effect of preemption on performance, the PBS scheduler is aware of the heterogeneity of the virtual domain and never schedules jobs on a mixture of virtual and physical machines. The application's ability to adapt is particularly important in our case, since virtual machines are created and destroyed on demand.

Demand Heuristic. In general, there is no requirement for how machine brokers calculate demand. For our prototype, we use the PBS scheduler's work queue as a measure of the demand on the domain. Each virtual domain a uses a PBS scheduler that multiplexes the set of physical machines P_a and the set of virtual machines V_a. The machine broker queries the PBS scheduler for the number and size of jobs j in the queue Q_a. The result is used to assess the current demand d_a, defined as the number of nodes required to satisfy all jobs $j \in Q_a$.

$$d_a = \sum_{j \in Q_a} j_{nodes\_required}$$

Borrowing and Lending. Based on the calculated demand d_a and current number of machines lent $|L_a|$ or borrowed $|V_a|$, the machine broker calculates the machines needed n_a.

$$n_a = d_a - [(|P_a| - |L_a|) + |V_a|]$$

The value of n_a determines if it is desirable to lend, borrow, or return previously borrowed machines. If n_a is positive, virtual domain a needs to acquire n_a nodes; if it is negative then virtual domain a can lend or return $|n_a|$ nodes; and if it is zero virtual domain a has exactly enough nodes to satisfy its own demand.

The lending policy implemented in our prototype allows up to half of the domain's idle nodes to be borrowed by other clusters. This allows substantial resources to be offered to other domains in quiet periods, while guarding against resource shortages during sudden spikes in demand.

Reclamation Technique. It is sometimes necessary for a virtual domain to recover machines from other clusters, due to sudden increases in demand. In these cases, machines are returned according to the reclamation policy specified in the lending contract.

In our prototype, machines are returned immediately upon request from the lending domain. Any jobs running on these machines will be terminated, and must restart. This recovery process is managed by the PBS scheduler. Clearly, such preemption has a negative impact on the throughput of

the system, and steps must be taken to minimize the cost of this interference. When possible, our scheduler runs jobs only on physical machines belonging to the virtual domain. Additionally, to minimize the overhead of preemption and improve the network locality of a job, the scheduler never schedules jobs on a mix of virtual and physical machines.

As an alternative, a gradual reclamation policy may be implemented. Future work on VioClusters will study policy interactions, most notably in the area of machine reclamation techniques.

4 Experiments

In this section we present several experiments that show the feasibility of VioCluster. First we measure several individual VioCluster system properties, then we show the results of a large-scale VioCluster simulation based on the prototype's measured behavior, using real workload traces from production clusters.

4.1 System Prototype Measurements

For the prototype measurements we used two clusters on Purdue's campus. One cluster is administered by the *nanoHub* as part of the NFS Network for Computational Nanotechnology (NCN) and the other cluster is administered by our laboratory in the Computer Science department. The NCN cluster is composed of dual processor 3.06GHz Intel Xeon machines with 2GB of RAM connected by 100Mb/s Ethernet. The Computer Science cluster is composed of 2.6GHz Intel Xeon machines with 2GB RAM connected the 100Mb/s Ethernet. The connection between the clusters is through Purdue's campus network. Although the UML-based experiments provided good results, our continuing work with Xen promises significantly increased performance.

Metric	Value
Execution Slowdown on VM	15%
VM Boot Time	40 seconds
VM Halt Time	16 seconds
VIOLIN Bandwidth Penalty	5-15%
VIOLIN Latency Penalty	5-10%

Bandwidth and Latency. From our previous work [7] we have observed that VIOLIN networks affect communication bandwidth and latency. VIOLIN decreases the bandwidth between machines by 5-15% and increases the latency by 5-10%.

Computation Overhead. We have found that computation and communication intensive workloads, such as High Performance Linpack (HPL), run 15% slower on VMs connected to a VIOLIN network [14]. Less communication intensive applications would experience a smaller decrease in performance.

Image Transfer, Boot, and Halt Times. Aside from runtime overhead, VMs require time to be setup and destroyed. By using small COW filesystems and transferring the base system images ahead of time, we can transfer the system image to a physical host in under a second. This short transfer time means that the VM boot time dominates the creation process. We measured boot times for two VM images: the first being a modified Fedora Core 1 server installation, and the second a minimal RedHat 8.0 system. The larger image took 40 seconds to boot on the NMI cluster, while the smaller took 5 seconds. Halt times were found to be 16 seconds and 3 seconds respectively.

4.2 Simulation

Simulation Setup. Evaluating VioCluster on real workloads was not practical, due to the difficulty of scaling workloads that originally ran in months or years to run in a reasonable period. We therefore developed a simulator that not only enables us to accurately evaluate the system on large workload traces, but also to simulate far larger clusters than we have available.

Our simulation takes into account our measured VM transfer, boot and halt times, as well as the computation and communication overheads of VIOLIN. The machine brokers use the preemptive trading policy described in section 3.2. Each broker calculates the virtual domain's demand every hour, and then takes action based on the results.

The virtual domain's size and usage pattern is created using traces obtained from production machines. The two traces used are publicly available (CTC and OSC from [21]) and are composed of a 512 CPU machine at The Cornell Theory Center (CTC) and a 178 CPU machine at Ohio Supercomputing Center (OSC). What follows is the results of the simulation, showing that by using our machine trading mechanism and policy the perceived processing power of both clusters is improved.

Observed Demand. Figure 3 shows the demand on each of two virtual domains over time with and without sharing enabled. The demand without sharing is what would be observed if the load patterns were submitted to independent clusters. It can be seen that the OSC cluster has two distinct spikes of very high demand and the CTC cluster has many spikes that are relatively small.

When the clusters are run with sharing enabled the virtual domains are able to handle demand more efficiently. With sharing, the large spikes of the OSC cluster are significantly reduced while the CTC cluster is relatively unaffected.

Machine Borrowing. Figure 4 shows the machines traded between domains over time. Positive numbers in-

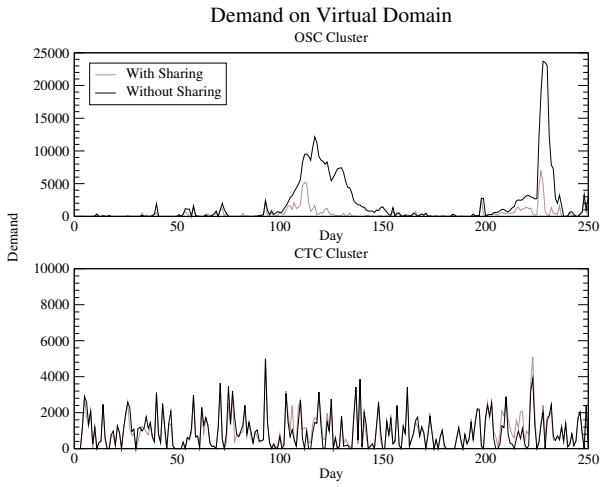


Figure 3. Demand on the virtual domain over time.

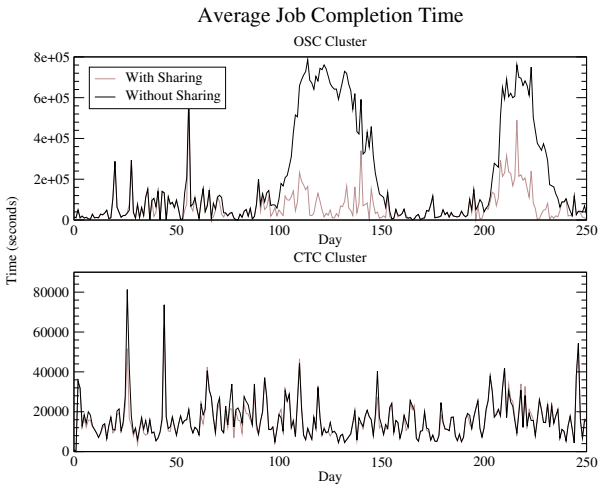


Figure 5. Average time from job submission to job completion of qualitatively different workloads.

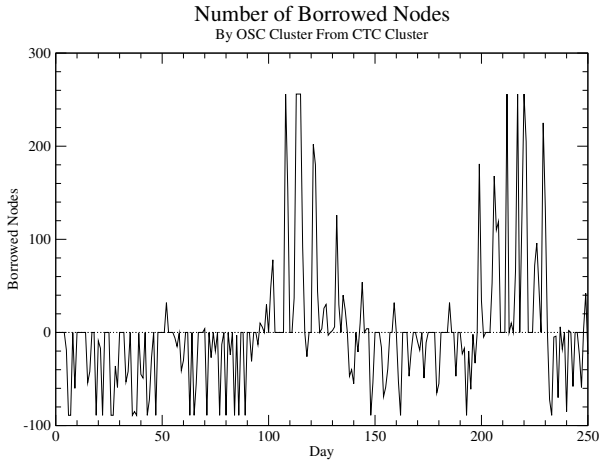


Figure 4. Number of borrowed nodes.

dicate that the OSC virtual domain is borrowing from CTC, while negative numbers indicate CTC borrowing from OSC. A correspondence can be seen between the areas of high demand in the OSC domain in Figure 3 and the number of machines borrowed from CTC. The high frequency of spikes in Figure 4 could be reduced by either a more conservative lending strategy or a more gradual reclamation policy.

Job Completion Times. Figures 5, 6, and 7 show the responsiveness of the system from a user's perspective. They show the average time between a job's submission to the cluster's queue and its completion both with and without sharing. Each figure depicts the interaction of cluster workloads with different qualitative properties.

Figure 5 shows the interaction between two different workload patters. The OSC cluster has two large spikes of high demand, while the CTC cluster has a steady pattern of small spikes. From the graph, we see that both clusters benefit from sharing, with the OSC cluster almost completely eliminating its spikes.

The reduction in completion time during peak demand is due to a reduced amount of time the jobs wait in the queue. Referring back to Figure 4, we see that during the period of high workload, the OSC virtual domain is often able to borrow a significant portion of the CTC virtual domain's nodes. The borrowed nodes allow the OSC virtual domain to run more jobs at once reducing the average completion time drastically.

Figure 5 shows occasional points where completion time increases when sharing is enabled. This is caused by jobs being preempted when a physical machine was reclaimed, and subsequently restarted. Techniques such as check-pointing, VM migration, or complementary advanced scheduling algorithms designed for dynamic systems, can reduce or eliminate this problem.

In Figure 6, two traces with corresponding spikes in demand were created by extracting sections of the OSC cluster trace. From the figure we see that if one of the virtual domains has a spike at the same time the other does not, the spike is largely mitigated through machine borrowing. On the other hand, if both virtual domains experience a spike at the same time, very little improvement is seen, however, neither cluster experiences a reduction in performance.

Figure 7, created from sections of the CTC trace, shows the opposite situation. Here neither cluster experiences a large spike in demand, and as a result neither benefits greatly from sharing. However, again, neither show a per-

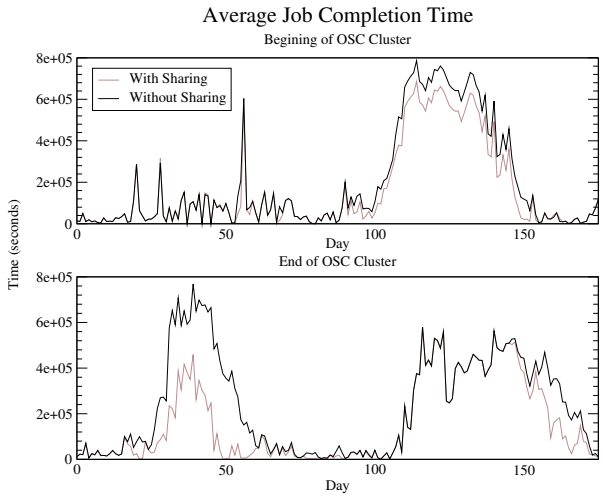


Figure 6. Average time from job submission to job completion of high demand workloads.

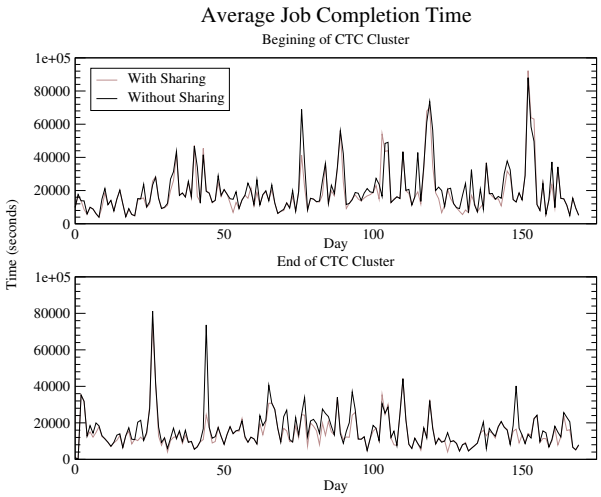


Figure 7. Average time from job submission to job completion of low demand workloads.

formance decrease.

From theses results we can conclude that by sharing computational resources through VioCluster great gains in the user's perceived performance are possible. As expected, clusters that are experiencing extreme spikes in workload benefit most from sharing. Unexpectedly, their gains do not penalize the donating clusters. In addition, these gains are the product of a relatively simple trading policy which demonstrates the the effectiveness of the system and the potential of more advanced policies.

5 Related Works

Currently, the most common resource sharing methods are seen in the creation of large, shared Beowulf [3] style cluster computers that multiplex resources through a batch scheduler such as PBS [11]. A common example of such systems would be large general use clusters administered by a campus-wide authority and used by members of many departments. More recent examples of resource sharing include cycle stealing applications such as Seti@Home [15], as well as meta-scheduling dedicated Grid infrastructures like Globus [6, 5], Condor [17], In-VIGO [1] and PUNCH [8], among others. 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 the problem that jobs are run on nodes over which the job owner has no control.

The system most similar to our own is Cluster-On-Demand (COD) [4]. Like VioCluster, COD allows dynamic sharing of resources between multiple clusters. In a similar fashion to Oceano [22] and Emulab [20], COD reallocates resources by using remote-boot technologies to clear a physical machine and 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. When we compare VioCluster with COD we see two projects that have very similar goals; however, COD works by reinstalling the base Operating System of the resources. VioCluster creates VMs running on top of the existing host OS. COD is more suited for homogeneous pools of machines supporting logical clusters administered by trusted local administrators. The authors of COD are also studying market-based algorithms for negotiating control over resources in a shared cluster environment [13]. We have similar goals with our machine broker policies, however we do not focus on market-based strategies.

Virtual networking is a fundamental part of our work. Available machine virtualization techniques do not supply advanced virtual networking facilities. UML [18], VMware [19] and Xen [2] all provide networking services by giving the VMs a real IP address from the host network. PlanetLab [12] uses a technique to share a single IP address among all VMs on a host by controlling access to the ports. These techniques allow VMs to connect to a network but do not create a virtual network. Among the network virtualization techniques are VIOLIN [7] and VNET [16, 10] which create virtual network overlays of VMs residing on distributed hosts. VNET provide a virtual a layer-2 Ethernet that connects remote VMs to a local physical LAN. VIOLIN creates completely isolated virtual environments

including virtual switches, routers, and VMs.

Another interesting use of VMs is In-VIGO [1]. In-VIGO is a distributed Grid environment supporting multiple applications which share resource pools. The In-VIGO resources are VMs. When a job is submitted, a virtual workspace is created for the job by assigning existing VMs to process it. During the execution of the job the VMs 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 [23]. Provided with In-VIGO is an automatic VM creation project called VMPlants [9]. VMPlants is used to automatically create custom root file systems to be used in In-VIGO workspaces. The VioCluster authors are currently participating in an In-VIGO deployment at Purdue as part of the NCN's *nanoHub*.

6 Conclusion

We have presented the design and implementation of VioCluster, a virtualization based computational resource sharing platform. Using VioCluster, independently administered computation domains can lend and borrow nodes, increasing utilization and reducing idle node time. We have implemented a prototype VioCluster system and have created a large scale simulation based on real prototype parameters. The results of the simulation, using real workload traces, show that VioCluster using relatively simple machine trading policies leads to potentially large increases in the perceived computational power of administratively autonomous clusters. At the time of this writing, we have completed an implementation of VIOLIN for Xen virtual machines. Our initial experiences with VIOLIN and Xen indicate improved performance over UML and show great promise for the future of VIOLIN and VioCluster.

7 Acknowledgments

We would like to thank the anonymous reviewers for their constructive comments and suggestions. We thank Sebastien Goasguen and Xuxian Jiang for their valuable support and input during this work. The NMI nanoHUB project provides the infrastructure to deploy and evaluate our prototype. This work was supported in part by NSF Grants SCI-0438246 and SCI-0504261.

References

[1] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu. From virtualized resources to virtual computing grids: The in-vigo system. *Future Generation Computer Systems*, 2005.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP 2003*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[3] http://www.beowulf.org.

[4] J. Chase, L. Grit, D. Irwin, J. Moore, and S. Sprenkle. Dynamic virtual clusters in a grid site manager. In *HPDC 2003*, June 2003.

[5] 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.

[6] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[7] X. Jiang and D. Xu. Violin: Virtual internetworking on overlay infrastructure. Technical report, Purdue University, 2003.

[8] N. H. Kapadia and J. A. B. Fortes. Punch: An architecture for web-enabled wide-area network-computing. *Cluster Computing*, 2(2):153–164, 1999.

[9] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo. Vmplants: Providing and managing virtual machine execution environments for grid computing. In *SC 2004*, page 7, Washington, DC, USA, 2004.

[10] J. Lange, A. Sundararaj, and P. Dinda. Automatic dynamic run-time optical network reservations. In *HPDC 2005*, 2005.

[11] http://www.openpbs.org/.

[12] V. Ramasubramanian and E. G. Sirer. The design and implementation of a next generation name service for the internet. *SIGCOMM Comput. Commun. Rev.*, 34(4):331–342, 2004.

[13] M. Ripeanu, M. Bowman, J. Chase, I. Foster, and M. Milenkovic. Globus and planetlab resource management solutions. In *HPDC 2004*, June 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] http://setiweb.ssl.berkeley.edu/.

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

[17] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The condor experience. *Concurrency and Computation: Practice and Experience*, 2004.

[18] http://user-mode-linux.sourceforge.net.

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

[20] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *OSDI02*, pages 255–270, Boston, MA, Dec. 2002.

[21] http://www.cs.huji.ac.il/labs/parallel/workload/.

[22] L. Zhang and D. Ardagna. Sla based profit optimization in autonomic computing systems. In *ICSOC 2004*, pages 173–182, New York, NY, USA, 2004. ACM Press.

[23] M. Zhao, J. Zhang, and R. Figueiredo. Distributed file system support for virtual machines in grid computing. In *HPDC 2004*, 2004.