

Maestro-VC: On-Demand Secure Cluster Computing Using Virtualization

Nadir Kiyancilar Gregory A. Koenig William Yurcik

National Center for Supercomputing Applications (NCSA)
University of Illinois at Urbana-Champaign
{*nadir, koenig, byurcik*}@ncsa.uiuc.edu

Abstract

On-demand computing is the name given to technology which enables an infrastructure where computing cycles are treated as a commodity, and where such a commodity can be accessed upon request. In this way the goals of on-demand computing overlap with and are similar to those of Grid computing: both enable the pooling of global computing resources to solve complex computational problems.

Recently, virtualization has emerged as a viable mechanism for improving the utilization of commodity computing hardware. This field has seen much research for potential applications in the field of distributed and Grid computing. In this paper, we present an architecture and prototype implementation for Maestro-VC, a system which takes advantage of virtualization to provide a sandboxed environment in which administrators of cluster hardware can execute untrusted user code. User code can run unmodified, or can optionally take advantage of the special features of our system to improve performance and adapt to changes in the environment.

1 Introduction

Recent years have seen a tremendous increase in the amount of computing power available in the form of commodity hardware. The trend to date in high performance and scientific computing has been to move away from special purpose vector computing machines and focus on solutions based on commodity parts. Clusters have proven to be a scalable and cost-effective solution to harnessing large amounts of computing power. However, for most small installations the maintenance of such a resource proves to be a less scalable endeavor. This effectively limits the manageable size of a cluster of machines and forces scientists to look elsewhere if more cycles are desired.

Grid computing, a concept first published in [10, 9], is the sharing of resources by task-oriented ‘virtual organizations’ which are dynamic in the sense that they are formed and exist solely for the purpose of solving a given computational problem, and no longer. The focus of ‘the Grid’ in this sense is as a medium for transparently sharing resources, in a manner which is globally convenient to all users of the shared resource pool, but which also obeys the local policies of each contributor to the pool.

While not contradictory to the goals of Grid computing, on-demand computing does have a different focus. An on-demand computing infrastructure also enables sharing of computing resources. However, the focus of this paradigm is on the provision of computing cycles as a uniform resource, and immediate access to this resource. In other words, computing power in an on-demand computing infrastructure can be viewed as a commodity (that of computing cycles), and this power is available in any reasonable quantity when users request it (like other public utilities, such as water or electricity). While Grid computing’s main focus is on the ability to easily coordinate and aggregate resources between willing parties, on-demand computing focuses on transparent access to and efficient utilization of these resources. Recent work at the University of Illinois and the National Center for Supercomputing Applications [19] uses the Faucets scheduler [16] as an enabler for on-demand computing. Faucets is a framework for developing scheduling solutions involving single clusters or federations of cooperating clusters. Developers extend the Faucets framework by creating *scheduling strategies* which implement resource scheduling policies. Our work involves writing scheduling strategies to, for example, leverage checkpoint/restart mechanisms within various runtime systems, create and destroy virtual machines dynamically, etc.

Faucets is designed to work closely with the Charm runtime system [15], which multiplexes message-driven distributed object called *chares* across the CPUs of a computational resource. Chares represent units of work within a

computation and may be transparently migrated at runtime, allowing load balancing of resources. Further, recent work has shown that Charm++ presents a compelling solution to several challenges in Grid computing such as overcoming the effects of latency or masking heterogeneity of interconnects, processor speeds, etc. [18]. Finally, a version of the popular parallel programming standard MPI, called Adaptive MPI [13], is built on top of the Charm runtime system and brings Charm features to more traditional MPI applications.

In this paper, we present Maestro-VC (VC==Virtual Cluster), a set of system software which uses virtualization to multiplex the resources of a computing cluster for client use. Because virtualization is used, the virtual machines presented to users look to application binaries exactly like real machines, allowing distributed code to run unmodified. Thus existing applications can immediately take advantage of our proposed virtualized cluster architecture. On the other hand, while the virtualization software used in our system is efficient in terms of the overhead vs. native execution on individual hosts, the mapping of virtual to physical resources over the network may result in certain inefficiencies. This is due to the loss of information available to a program about its actual physical environment, and to the corresponding lack of information about program behavior available to cluster management software running outside a virtual cluster. In other words, we believe (though have not yet verified) that distributed virtualized systems will exhibit problems analogous to those of single-system virtual machines (VMs), for example the double paging mentioned in [25]. The solution to this problem is analogous to that used in the single system VM case: allow information exchange between the VM and virtual machine monitor (VMM) to improve performance. In our design, scheduling is split into two levels, with an upper level scheduler managing the allocation of VMs (also referred to as ‘slices’ of machines in other works), and an optional low-level scheduler supplied by users which manages the use of VMs inside one of these global allocations. The high-level scheduler implements a protocol for resource negotiation, allowing client code to borrow more resources if necessary, and in general to adapt to changing conditions on the physical cluster. These changes can be due to machine state (impending machine faults are detected and program state should be saved) or global decisions (the cluster decides to reclaim some nodes from a virtual cluster to give them to a higher priority job).

The nature of on-demand computing is anonymous untrusted users executing processes on unknown and untrusted resources. Potentially these anonymous users may gain access to and misuse resources from resource providers. Virtualization can be a way to enhance protection for this security concern by isolating physical resources from direct access by users.

The contribution of this paper is the presentation of a virtualized cluster management environment with a unique combination of features relative to other systems. While other works exist which have implemented parts of the system we have described, we believe we are the first to offer a virtualization based system which incorporates information exchange between virtualized and native environments via two-level scheduling. Furthermore, no other system offers the tradeoff between portability and adaptability of virtualized applications that our system does by making local scheduling completely optional.

The remainder of this paper is organized as follows. Section 2 surveys related work. Section 3 presents the system architecture of Maestro-VC. Section 4 describes our initial implementation efforts (more detail on our implementation will be included in the final version of this paper). We end with a summary and conclusions in Section 5.

2 Related Work

On-demand computing refers to on-demand access to resources – either immediate access, or in the case of over-subscribed resources, access based on consumer job priorities or job deadlines. With physical resources, this can be accomplished via checkpoint/restart: low priority jobs can then be checkpointed to make resources available to higher priority tasks. As hinted at in Section 1, this is usually accomplished at the application level as in systems such as Condor [1]. Alternative system-level solutions exist such as Cluster-on-Demand [22] and the commercial Oceano¹. These systems perform scheduling and allocation at the physical level by allocating physical machines to customers. Oceano and Cluster-on-Demand (COD) achieve flexibility by allowing machines to be re-imaged with any compatible OS. Absolute control over all resources must be done at the BIOS and hardware level in both systems. For example, both require special hardware to enable remote power management, so that misbehaving nodes can be rebooted forcibly.

Another way of addressing the problem of on-demand computing is to use virtualization to multiplex virtual resources onto a smaller set of physical resources². In a manner analogous to Cluster-on-Demand, our system can

¹which the authors of [22] in fact note as an inspiration

²For a survey of the applicability of virtualization methods to on-demand computing, see [17]

boot virtual machines on-demand on physical machines. Full control over guest machines is maintained as in COD. The difference is that customer ‘machines’ are now VM instances, and control is enforced at the VMM layer rather than the physical. It should be noted that COD does not address on-demand computing; the required fine-grained access to computing resources, not possible with COD’s whole-server allocations, is precisely what we are achieving by subdividing physical resources into virtual ones.

Research in virtualization has long and rich history, beginning with IBM mainframes in the 1960s [11]. Systems such as IBM’s System/370 [24, 6] have characteristics to allow CPU and I/O virtualization to be carried out in hardware³, while most commodity architectures including the popular IA-32 lack this support⁴. To address the shortcomings of these architectures, two virtualization techniques have emerged: full-system virtualization using binary translation, and paravirtualization.

Full system virtualization on non-virtualizable architectures as defined in [23] involves running unmodified guest operating systems in unprivileged CPU mode so that privileged machine state never leaves the control of the VMM. Virtualization differs from instruction by instruction emulation in that as far as possible, native hardware is used to execute guest instructions. The exception occurs when in guest OS ‘privileged’ mode: in this case binary translation is used to intercept problematic ‘privileged instructions’ and replace them with safe calls into the VMM. VMWare [3] is the most popular commercial example of this technique. A drawback of systems such as VMWare is the high overhead: while native execution speed is achievable in CPU-intensive virtualized code, guest VM applications which execute frequent system calls, and which perform a large amount of I/O can experience significant slowdowns. Advanced techniques are required to lessen the performance hit VMs experience in these cases [25].

Paravirtualization uses a different method to address the same problem of trapping privileged guest VM instructions. In this case, the guest OS is modified at the source level to replace privileged OS calls with explicit jumps into a VMM. Paravirtualized systems thus sacrifice the transparency of fully virtualized systems in varying degrees for the sake of more efficient utilization of resources on non-fully virtualizable architectures. The most notable systems are Denali [26] and Xen [8], which is in fact used in the current implementation of our system. User-Mode Linux [7], a port of Linux which runs as a user space process on a native Linux system, can also be viewed as an example of a paravirtualized system, but has severe performance shortcomings for syscall or IO-intensive guest applications.

A number of other researchers have investigated systems to enable sharing of distributed computing resources for the purposes of Grid and On-demand computing. In-VIGO [4] is a system for enabling ‘virtual computing grids,’ which build on virtualized resource management mechanisms to present virtualized ‘grid sessions’ to users. In-VIGO encompasses a stack of software whose scope is larger than that of our own work, but the underlying virtual machine management software, VMPlants [20] is similar to our own VM creation facility. SODA [14] is a virtualization based system which targets application service providers. SODA uses virtualization in the form of User-Mode Linux. The Xenoserver platform [12] is a system from the creators of Xen which, like our own system, builds on the Xen VMM. Xenoserver’s end goal is the establishment of an economy for public utility computing, which is similar to the goals of on-demand computing. The Globus Workspace Management Service provides for abstracted ‘execution environments’ which can be made available to authorized users on-demand. These execution environments are called *virtual workspaces* [2]. A virtual workspace can be implemented in a number of ways, including a dynamically generated UNIX account or a virtual machine.

Development previews of the Globus Management Service focus on workspace encapsulation using Xen virtual machines. Xenoserver is intended to run on a number of independent physical machines; current implementations of the Globus Workspace Manager also work with individual physical machines⁵. Our work has a slightly different focus in that we focus from the start on the physical cluster as the atomic unit of resource management: all requests for machines go through a cluster scheduler and result in an allocation of VMs under that scheduler’s control.

3 System Architecture

In this section we lay out the high-level design of Maestro-VC and describe the role of each of its software components. A physical overview of the Maestro-VC architecture is shown in Figure 1. Each machine runs a virtual machine monitor. Nodes allocated to on-demand clients are guest VMs instantiated on a compute node. To extend the single-system example, the collection of VMMs under the control of the login node daemons is a cluster-wide VMM.

³a formal statement of these theories is found in [23]

⁴although hardware virtualization support in the form of Intel’s Vanderpool is forthcoming in the first half of 2006, and AMD’s Pacifica shortly thereafter

⁵Support for clusters is planned

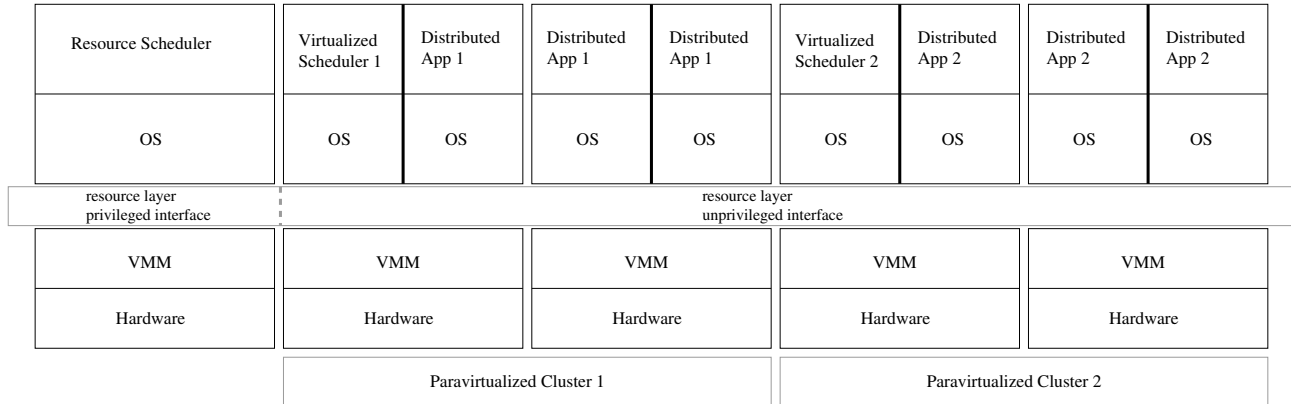


Figure 1: Physical View of Maestro-VC Architecture

Taking this example further, virtualized clusters are analogous to VMs in the single-system case. A list of the logical components of Maestro-VC and a short description of each follows:

- **Gateway** - The Gateway is the interface presented to clients wishing to run jobs on a Maestro-VC-enabled cluster. All client interaction before acquiring an allocation is through the Gateway.
- **Stager** - The Stager is responsible for setting up a new virtual cluster once the Global Scheduler has deemed this appropriate. This requires coordinating between the Allocator and Node manager to implement the setup, and then informing the Global Scheduler so the job can start.
- **Global Scheduler (GS)** - This scheduling component implements global scheduling policies. These are site specific and may be tailored to increase profit, throughput, etc.
- **Allocator** - The allocator is specifically responsible for disk allocation during the job staging process, and is thus a functional subcomponent of the Stager.
- **Node Manager (NM)** - From a node perspective, most command traffic in Maestro-VC follows a master-slave pattern, from the head or login node to compute nodes. The Node Managers are the entities which realize most of the commands handed down from one of the components above on each local node.
- **Local Scheduler** - This optional component is specified by a user running a virtualized cluster job, and can negotiate with the GS to respond to changes in allocation or other events.

Figure 2 shows the location of the Maestro-VC software components as installed in a cluster. The Gateway, Global Scheduler, Stager and Allocator run on the login node in a physical cluster, while a Node Manager instance exists on each compute node. To further illustrate the Maestro-VC system, we give a detailed description of the interactions required to allocate a new ‘virtual cluster.’

The NMs, as illustrated in the example below, can be invoked by administrative processes on the master node in order to reclaim nodes, create new allocations, etc. A restricted subset of the NM commands can also be invoked by an LS local to each virtualized cluster. Conceptually, the NMs make up an abstract control resource control layer (the virtualized resource abstraction layer, or VRAL), through which all access to physical computing resources is mediated. All access to the VRAL is through the master node in a cluster; either one of the master node daemons initiates a control function, or the LS invokes a function through the GS. As in Figure 1, this means the VRAL appears to the entire cluster and all virtualized clusters running on it as a single entity.

At a high level, the GS allocates a group of whole and partial physical machines as VMs. In the intended design, each distinct group of VMs is placed under the control of a different client, and is isolated on its own restricted subnet. The VMs appear just as ordinary physical machines to users, who have root login access and the option to install any additional software on the VMs. At this point a normal distributed job, such as an MPI job, can be started on the machines. Performance benefits may be realized by taking advantage of Maestro-VC’s bi-level scheduling. Bi-level scheduling refers to the interaction between the GS, and a LS running in virtualized context. The LS establishes a

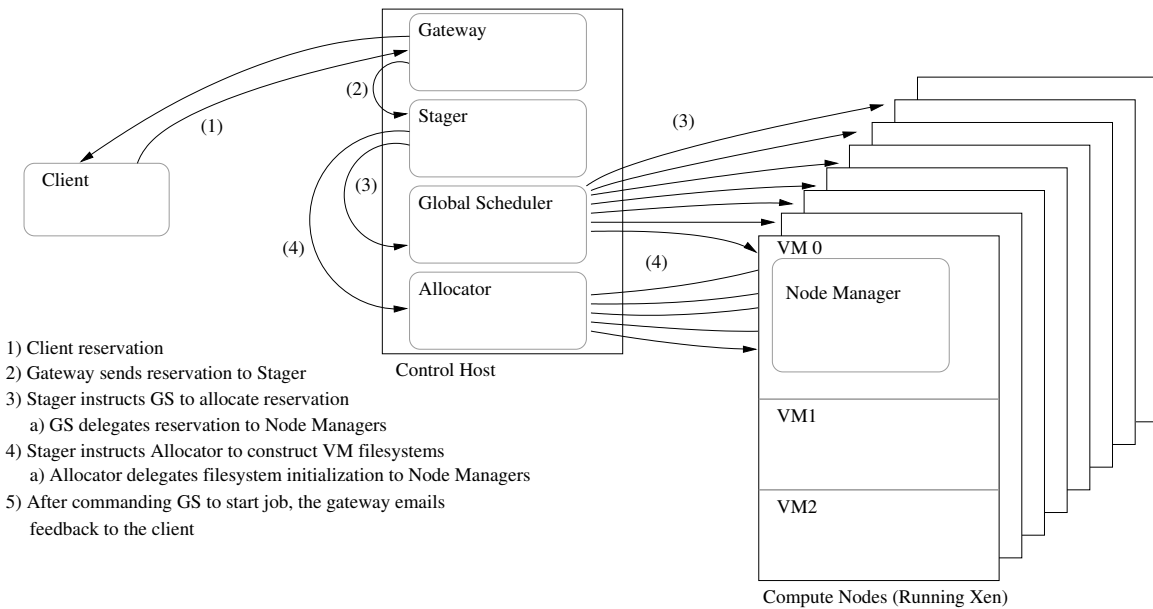


Figure 2: Logical View of Maestro-VC Architecture

connection (via Berkeley sockets) with the GS. In effect, this gives the LS access to the restricted resource management interface described above. The restricted interface can be used to request more VMs, which can be granted if available. Conversely, this connection can be used by the GS as a callback mechanism of sorts: upon a change in cluster conditions, most likely brought about by the arrival of new jobs, the GS may decide to reallocate some nodes to a higher priority job. Without the ability to send feedback to the running virtualized job, the GS can only forcibly reclaim the nodes of the lower-priority job⁶. If informed by the GS, however, an LS could take responsive action to this environmental change. This could entail either triggering an application-level checkpoint of the running virtualized application, or the resizing of the virtualized job to fit the new constraints imposed by the scheduler. The virtualized job, though slowed down, can continue to run. Thus through GS-LS interaction, higher throughput can be maintained on a cluster at the expense of some loss of transparency.

Another key feature of Maestro-VC is that the GS can be quite simple. By delegating adaptive features to the Local Schedulers running in each virtualized cluster job, the GS is free to resize allocations as needed to optimize site-specific priorities, such as throughput, profitability, etc.

Figure 2 highlights a functional example of an interaction between Maestro-VC components – the steps in allocating a virtual cluster to a client. The following is a detailed explanation of the figure. Bracketed numbers refer to labels in Figure 2.

1. A client queries the Gateway for availability. If the request can be satisfied within the given time (determined by a query to the GS), the Gateway will respond in the affirmative, and the client can then submit a job.
2. The client submits a job reservation to the Gateway(1). This reservation consists of the description of the number, type, and computing power of nodes requested. In addition, initialization information and an optional URL from which to pull staging information is provided. If the job request is successful (the same feasibility check is made as before) the client will receive a token consisting of information on the number of nodes that have been allocated, and on information needed to log into these nodes. The Gateway then submits this job to the Stager(2).
3. The Stager contacts the GS (3) and informs it to make room for the new allocation. The GS clears memory and CPU allocations sufficient for the new job by either using free physical nodes, or by resizing the memory allocations and CPU reservations of other jobs. Sufficient disk space must also be cleared to house the new

⁶Unlike in a non-virtualized system, the scheduler can make no assumptions about what kind of applications are running on the VMs under its control.

VMs' virtual disks. As the GS runs on a dedicated physical node, these requests are delegated to the NMs running on each compute node(3).

4. After the GS has completed its reservation of resources for the new job, the Stager hands off to the Allocator(4) information on what type of file system should be initialized for the new VMs to be started. Again this initialization is delegated to the NMs(4).
5. Finally, Stager informs the GS to start the new jobs. The GS informs the NMs on each compute node to boot the new VMs and configure virtual networking properly, and records the job start in its scheduling algorithm. Finally user-specific staging data is pulled and initialization actions are run. It is during the initialization step that a client can specify the startup of an LS to interact with the GS as described above. At this point an email is sent to the client informing them that the resource is ready and available for access.

4 Implementation

Our current prototype is implemented on top of the Xen virtual machine monitor, along with a number of daemons running in privileged Xen 'domain 0' VMs on the machines of a cluster. The intended structure of a cluster running Maestro-VC is as follows: one node is dedicated to cluster management through the Gateway and GS, and the other physical machines run the the Xen VMM, with a NM daemon running in the privileged domain 0 of each. While it is not strictly necessary to have the master node run virtualized at all (no jobs are executed on this machine, and the only communication with virtualized cluster jobs is through sockets), we do this in the interest of a more uniform machine setup across nodes.

The various daemons which make up our prototype are currently implemented in the Python programming language. As detailed in section 3, most of the actual VM setup and execution is delegated by programs on the master node to the NMs on the compute nodes. The daemons comprising the functionality of the NMs in turn interact with the Xend control daemon running on each compute node to realize the commands handed down to them.

Currently, job queries and requests are submitted to the job Gateway in a simple XML-based format, with all client-Gateway communication handled via SOAP requests over HTTP transport. In addition, all internal communication in the Maestro-VC architecture also uses SOAP.

4.1 Disk Allocation

A number of the other systems mentioned in related work must deal with the problem of disk imaging for VM allocation. Other works have shown that depending on network and disk bandwidth, imaging and configuring the virtual disk for a VM can take anywhere from one to ten minutes [5, 22]. Like other dynamic cluster allocation schemes, we feel that users will be able to accept a waiting time of minutes to initialize a job which will most likely run for a few hours. Currently, there is no distributed disk initialization functionality present in Maestro-VC: Upon delegation by the Allocator, all nodes locally reinitialize one of their 'guest' partitions, and locally uncompress one of a number of disk images into it. From here local customization (editing configuration files for the network and host name, for example) is done outside of guest 'context' by mounting the guest partition in the local domain 0 and performing the necessary operations. Finally the guest file system can be unmounted and a VM started with it as root. Caching of disk images on local nodes provides optimization opportunities when dealing with short jobs, or checkpointed jobs with a short execution time remaining. In order to avoid excessive 'VM flap' in such cases, the scheduler can wait until after another job using the same file system type (e.g. 'Debian Linux 3.1') has run before activating short jobs. A low execution overhead can then be insured for shorter jobs relative to their overall execution time.

Other systems also have mechanisms for initializing guest file systems from the master node. This is a more space efficient solution, as each image must be stored only once on the master node, and can be copied to the compute nodes on demand, obviating the need to store a number of such sample file systems on each compute node. Various schemes exist for the task of copying this large amount of data to each node, such as Frisbee [21]. Whereas our initial implementation of this functionality will use a linear copying scheme (as in [22], for example), we will supplement our existing functionality by caching recently used images on the local nodes in order to improve initialization speed for commonly used virtualized OSs.

4.2 Network Isolation

Our physical setup is an ‘isolated cluster,’ with the only physical connection to an outside network through the login or head node which does Network Address Translation for internal nodes⁷. Since guest VMs use virtual devices, it is possible to specify the MAC addresses of such devices, and by dedicating a range of MACs to each distributed job, to assign a contiguous range of IPs to each job. We use this method instead of VLANs to assign address ranges to jobs in Maestro-VC. Routing rules in the privileged domain 0 on each compute node can be used to enforce the restriction that one virtual cluster cannot access the address range of another, without the extra overhead involved in completely virtualizing the address space assigned to each. This strategy of fully controlling the access of each virtual cluster while making virtual nodes aware of their true address⁸ is an extension of paravirtualization to the networked environment.

5 Conclusion

We have presented the design of Maestro-VC, a virtualized cluster management system, and have outlined the implementation of a prototype. We feel that Maestro-VC is a first step toward a system to enable on-demand computing. The strict control of CPU and disk allocations allowed by a VMM are important to guaranteeing the quality-of-service experienced by virtualized jobs, as well as for system security. Further, by being able to provide varying degrees of support to any distributed job, not just those specifically designed for a particular API, our system becomes more generally usable. We hope that this generality will translate into increasing popularity as our system matures.

Future plans for Maestro-VC are to improve the maturity of the system and to move to a realistic testing platform. We have recently acquired two 32 node rack-mounted computing clusters, each equipped with dual processor 1U machines, and each equipped with high-performance Myrinet interconnects. We plan on publishing the job setup times and native vs. virtualized cluster performance as part of our next work on this Maestro-VC.

An additional direction we would like to see Maestro-VC move in is that of higher transparency to virtualized jobs. One example of this is migration: by having all VM instances mount their file systems via a high-performance network file system, seamless live VM migration can be performed between Xen systems on separate physical machines. This would allow us to resize the physical allotments of a virtualized job even if a Local Scheduler is unavailable on it.

References

- [1] The Condor Project. <http://www.cs.wisc.edu/condor/index.html>.
- [2] Globus Workspace Management. <http://workspace.globus.org/>, 2005.
- [3] VMWare. <http://www.vmware.com>, 2005.
- [4] 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*, April 2005.
- [5] K. Applby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger. Oceano – SLA Based Management of a Computing Utility. In *Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management*, 2001.
- [6] R. J. Creasy. The Origin of the VM/370 Time-Sharing System. In *IBM Journal of Research and Development*, September 1981.
- [7] J. Dike. User-Mode Linux. In *Proceedings of the 5th Annual Linux Showcase and Conference*, November 2004.
- [8] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the Art of Virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*, October 2003.
- [9] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure Working Group. Global Grid Forum (GGF), June 2002.
- [10] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.
- [11] R.P. Goldberg. Architecture of Virtual Machines. In *Proceedings of the AFIPS Computer Conference*, July 1973.
- [12] S. Hand, T. Harris, E. Kotsovinos, and I. Pratt. Controlling the Xenoserver Open Platform. In *Proceedings of IEEE Conference of Open Architectures and Network Programming (OPENARCH)*, 2003.

⁷although NAT is not strictly necessary

⁸notwithstanding the NAT performed in our installation

- [13] C. Huang, O. Lawler, and L.V. Kale. Adaptive MPI. In *Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing*, 2003.
- [14] X. Jiang and D. Xu. SODA: a Service-on-Demand Architecture for Application Service Hosting Utility Platforms. In *International Symposium on High Performance Distributed Computing (HPDC)*, 2003.
- [15] L.V. Kale and S. Krishnan. CHARM++ : A Portable Concurrent Object Oriented System Based On C++. In *Proceedings of the Conference on Object Oriented Programming Systems, Languages and Applications*, September–October 1993.
- [16] L.V. Kale, S. Kumar, J. DeSouza, M. Potnuru, and S. Bandhakavi. Faucets: Efficient Resource Allocation on the Computational Grid. In *International Conference on Parallel Processing (ICPP)*, Quebec, Canada, August 2004.
- [17] N. Kiyancilar. A Survey of Virtualization Techniques Focusing on Secure On Demand Cluster Computing. ACM CORR Technical Report cs.OS/0511010, November 2005.
- [18] G.A. Koenig and L.V. Kale. Using Message-Driven Objects to Mask Latency in Grid Computing Applications. In *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, April 2005.
- [19] G.A. Koenig and W. Yurcik. Design of an Economics-Based Software Infrastructure for Secure Utility Computing on Supercomputing Clusters. In *12th International Conference on Telecommunication Systems Modeling and Analysis (ICTSM)*, July 2004.
- [20] I. Krsul, A. Ganguly, J. Zhang, J. Fortes, and R. Figueiredo. VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing. In *Proceedings of IEEE/ACM Supercomputing Conference (SC)*, 2004.
- [21] M. Leigh. Fast, Scalable Disk Imaging with Frisbee. In *USENIX*, 2003.
- [22] J. Moore, D. Irwin, L. Grit, S. Sprenkle, and J. Chase. Managing Mixed-Use Clusters with Cluster-on-Demand. Duke University Department of Computer Science Technical Report, 2002.
- [23] G. Popek and R. Goldberg. Formal Requirements for Virtualizable Third Generation Architectures. *Communications of the ACM*, 17(7), July 1974.
- [24] L. H. Seawright. VM/370 - A Study of Multiplicity and Usefulness. In *IBM Systems Journal*, January 1979.
- [25] C. Waldspurger. Memory Resource Management in VMWare ESX Server. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.
- [26] A. Whitaker, M. Shaw, and S. Gribble. Scale and Performance in the Denali Isolation Kernel. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.