2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing

Saline: Improving Best-Effort Job Management in Grids

Jérôme Gallard, Christine Morin
INRIA Rennes – Bretagne Atlantique
Rennes, France
firstname.lastname@inria.fr

Adrien Lèbre
EMN, INRIA Rennes – Bretagne Altantique, LINA
Nantes, France
adrien.lebre@emn.fr

***Abstract*—Virtualization technologies have recently gained a lot of interest in Grid computing as they allow flexible resource management. However, the most common way to exploit grids relies on dedicated services like resource management systems (RMSs) to get resources at a particular time. To improve resource usage, most of these systems provide a best-effort mode where lowest priority jobs can be executed when resources are idle. This particular mode does not provide any guarantee of service and jobs may be killed at any time by the RMS when the nodes they use are subject to higher priority reservations. This behaviour potentially leads to a huge waste of computation time or at least requires users to deal with checkpoints of their jobs.**

In this paper we present `Saline`, a generic and non-intrusive framework to manage best-effort jobs at grid level through virtual machines (VMs) usage. We discuss the main challenges concerning the design of such a grid system, focusing on VM snapshot management and network configuration. Results of experiments show our proposal ensures an efficient execution of best-effort jobs through the whole grid.

An extended version of this paper can be found in [1].

***Keywords*-Saline; Virtualization; Grid Resource Management; Best-Effort Jobs**

I. Introduction

Grids (federation of clusters generally interconnected by a Wide Area Network – WAN) are used for a wide range of applications providing high-performance computing and large storage capacity. Although flexibility usage has been improved (deployment of dedicated environments [2], [3], lease concept [4], ...), the most common way of exploiting such distributed architectures still relies on a reservation scheme where a *static* set of resources is assigned to a job (or a user) during a bounded amount of time. This model of using grids leads to a coarse-grain exploitation of the architecture since resources are simply reassigned to another job/user at the end of the slot without considering the real completion of applications. In the best case, the time-slot is larger and resources are simply under used. In the worst one, running applications are withdrawn from their resources, potentially leading to the loss of all performed computations and requiring to execute once again the same request. If the former case is not really critical from the user point of view, the latter requires to deal with checkpointing issues to ensure the progress of the jobs. This is particularly true for the best-effort mode available in most resource management systems (RMS) where idle resources are assigned to users without any guarantee of service or time allocation. To deal with such issues, particular checkpointing mechanisms have been included in some RMSs [5], [6]. However, their methods are strongly middleware or OS dependant as they require either to link applications with dedicated checkpointing libraries or to exploit a checkpointing capable OS. Such constraints limit the locations where the application can be restarted since grids are generally heterogeneous environments. Consequently, best-effort job series are limited since users have to setup complex systems to periodically save results or to automatically resubmit the lost jobs [7].

In this paper, we propose to develop a generic and non-intrusive framework relying on virtualization snapshotting capabilities to efficiently manage best-effort jobs in coordination with any RMS at grid level. Several works show the interest for virtual machines (VMs) in the context of clusters as they provide flexible, isolated and powerful execution environments [8], [9]. Designing similar techniques at grid level implies facing new challenges [10]. For instance, VM migration in clusters usually relies on a SAN/NAS file system to share images. This particular way of sharing VM images cannot be exploited in grids and dedicated mechanisms have to be designed. Keeping in mind these constraints, we developed a first prototype entitled `Saline`.

Thanks to our proposal `Saline`, best-effort jobs can be transparently submitted into VMs so that the computation can be relocated in another location in the grid each time the resources have been taken away by higher priority jobs. Such an approach results in better performance concerning the total execution time of best-effort requests and a large benefit according to power consumption.

The remainder of the paper is organized as follows: Section 2 briefly presents the benefits of the latest VM capabilities in a grid context. Section 3 is dedicated to the VM management challenges. Section 4 presents an overview of our prototype, `Saline`, and discusses several experiments focusing on internal mechanisms. Related work is addressed in Section 5. Finally, Section 6 concludes and gives several perspectives.

An extended version of this paper can be found in [1].

II. Motivations

VM technologies provide flexible and powerful execution environments, offering isolation, security and snapshotting mechanisms, customization and encapsulation of entire application environments. Moreover, they allow bare hardware to be strongly decoupled from system software

1066-6192/10 $26.00 © 2010 IEEE
DOI 10.1109/PDP.2010.61

which is a predominant feature in the context of distributed architectures such as grids. Grids are composed of several resources exploited concurrently by multiple users. In addition, they are heterogeneous, geographically distributed and can potentially belong to distinct administrative areas. In this particular context, VMs can be exploited to encapsulate jobs and then make grid resource management more flexible: *the challenge of managing applications on grids is moved to the problem of managing VMs on grids.*

III. Transparent VM Management at Grid Level

Keeping in mind that we want to design a generic and non-intrusive framework (it should be able to run without requiring any modifications of the target RMS), we established several constraints to be able to submit and manage best-effort jobs until completion through transparent encapsulation into VMs. In our case, a typical scenario of best-effort job consists of an application potentially spread over multiple VMs communicating with each other at cluster level. Thanks to VM capabilities, it is possible to relocate a set of VMs in a coherent network state, from one site to another, assuming we use a reliable network protocol (*e.g.* TCP) and all the VM snapshots are done before fatal timeouts [11].

In this paper, we discuss two major points that we identified as challenges at the grid level: (i) the storage management of VM images during the best-effort job execution and (ii) the network configuration and mobility.

A. VM Storage Management

The VM storage management mainly concerns the way in which physical nodes access to the VM images. If diskless approaches (based on SAN or NAS solutions) can be exploited at cluster level to make VM image management easier, new strategies have to be proposed at grid level. In this paper, due to grid constraints, we consider that VM image is stored on the local hard drive of the node where the VM is currently running and we focus on the snapshot of the VM images when the RMS is going to withdraw some resources.

Saving of VM Snapshots: Various strategies could be implemented for saving the VM snapshots, *e.g.*, saving them locally or on a remote node. Each method having its own strength and weakness, in this paper, we focus on the remote method *i.e.* saving all the VM snapshots on a dedicated repository. In addition, we consider the use of Copy-On-Write techniques [12] which consist in saving only each VM image modification instead of each whole VM image.

Resolving the snapshot management issue consists in providing required mechanisms to be able to relocate a best-effort job when allocated resources are going to or have been withdrawn. According to RMS features, we have to design two approaches. The first one exploits notification callbacks whereas the second one makes periodical snapshots.

Using notification callbacks: Some RMSs provide callbacks to inform users when the slot allocated for a particular best-effort job is going to be withdrawn. When such an event occurs, our proposal should request the RMS for new resources on the grid. If resources are found (i), all the VMs involved in the job should be migrated on the new resources. Otherwise, all the VMs should be suspended to the VM repository (ii).

Taking into consideration these two cases, the challenge consists in freeing the impacted resources in a minimum amount of time. If the case (i) can be resolved by directly copying each image from the source node to the destination one, the case (ii) requires an advanced mechanism to efficiently copy the snapshots from the n nodes to the repository one. Such copies creates a network bottleneck at the master side that increases the required time for freeing resources. In addition, *naive* copies do not consider the issue of freeing each node as fast as possible *i.e.* some nodes of the best-effort job can be removed from the reservation and thus allocated to a higher priority job. In other words, we have to (i) reduce the whole time as much as possible and (ii) to free each resource as fast as possible.

To do this, we propose the following algorithm:
While there is a node waiting to transfer a VM image to the repository server:

- *copy one VM image from the node containing the lowest number of VM images*[1] *to the repository.*
- *from each other node, copy VM images to another node except if:*
  - *the destination node is already sending or receiving a VM image,*
  - *the destination node has less VM images than the sender node,*
  - *the sender node is already sending or receiving a VM image.*

Periodical snapshots: Considering that our framework should not require any modifications of the RMS and that some RMSs are not able to notify before removing best-effort jobs, we have to periodically make a snapshot of each VM to be able to restart them when resources have been withdrawn.

B. Network Configuration and Mobility

The network configuration of the VMs needs to be done considering the fact that our framework should manage best-effort cluster jobs at grid level, *i.e*, if needed, the set of VMs running a best-effort job could be migrated from one site to another transparently from the best-effort job point of view. This operation consists in dynamically assigning unique MAC and IP addresses to each VM in order to prevent any conflict with the physical infrastructure or with other VMs.

To solve this issue we propose to *deploy a DHCP-like server per site.* In this approach, there is one master node per site. This master is in charge of assigning distinct

[1]This could be easily extended to consider the whole size of VMs instead of the number.

MAC and IP addresses to all VMs belonging to the best-effort job in order to put all of them in a same subnet. In addition, the master should ensure this subnet is not already in use by other jobs. These requirements imply that, the master needs to send to each VM of the job, a *vmID* (VM identifier) and a *subnetID* (subnet identifier). To do that, we define a *networkmapID*, a bitmap of 65536 entries (16 bits) setting up on the master node. Each entry corresponds to a unique *subnetID* enabling to configure a unique subnet for a set of VMs. When a new best-effort job is launched, each VM retrieves a *subnetID* and a *vmID* from the master node.

MAC address configuration: MAC addresses are composed of 48 bits. Each NIC has a MAC address in which the first 24 bits (MAC1) are used for the manufacturer ID and the last 24 bits (MAC2) are unique to the NIC. In our case, we define our own manufacturer ID for MAC1. MAC2 ($\approx$ 16 millions of distinct addresses) is used to distinguish each VM NIC. We compose the first 16 bits of MAC2 with the *subnetID* received from the master. The last 8 bits of MAC2 are set according to the *vmID*.

48 bits		
24 bits MAC1: manufacturer ID	24 bits MAC2: unique to the NIC	
	16 bits *subnetID*	8 bits *vmID*

MAC address bitmap composition.

IP address configuration: Each VM statically sets its network configuration according to the two IDs received from the master. We assume the target grid platform allows the use of the 10.0.0.0/8 network. IP addresses are composed of 32 bits. In the 10.0.0.0/8 network, the first 8 bits (IP1) are assigned by the IANA Authority[2]. The second 24 bits are used by our framework. In these 24 bits, the first 16 bits (IP2) are set to the *subnetID* and the last 8 bits (IP3) are set to the *vmID*. This allow our framework to choose between 65536 networks (from 10.0.0 to 10.254.254) and 254 VMs (from 1 to 255). Of course, if the number of VMs is greater than 254, it should be possible to aggregate several subnets.

32 bits		
8 bits IP1: assigned by IANA	24 bits (used by our framework)	
	IP2: 16 bits *subnetID*	IP3: 8 bits *vmID*

IP address bitmap composition.

In our approach, one master node is required per site. That means the *networkmapID* bitmap should be shared between all the master nodes. This could be done by using a central server. However, such approaches lead to Single Point Of Failure issue and thus we prefer to use a distributed approach. Indeed, in our case, the number of requests made at the same time on the *networkmapID* is low. This allows us to use a strong consistency protocol on the read and write accesses to the *networkmapID*.

IV. Implementation and Experiments

Based on the previous analysis, we implemented a prototype, entitled `Saline`, which aims at solving the best-effort issue in Grid'5000 (G5K). G5K is a highly reconfigurable, controllable and monitorable experimental Grid platform gathering 9 sites geographically distributed in France featuring a total of 5,000 processors. The G5K resource management system relies on the OAR batch scheduler [13] and the `Kadeploy` software [2] which enables to deploy any user environment directly on bare hardware. In our experiments, `Saline` interacts with a non-modified version the OAR grid scheduler.

A. Implementation

From the implementation point of view, `Saline` is instantiated on each site. Each `Saline` instance relies on two major elements: the best-effort wrapper and the job manager. The first one provides a dedicated API to submit best-effort jobs and to specify execution constraints (VM image to exploit, hardware constraints, ...). The second one is in charge of VM management, periodically: (i) making snapshots of the current running VMs and saving them on the master node (i.e, the image repository), and (ii) asking the OAR service for the job status. When it detects that one job has been killed it launches the process of restoring the corresponding VMs: it submits a new best-effort request to OAR. If OAR is not able to find available resources on the whole grid, `Saline` waits a few minutes before asking OAR again. If OAR finds new resources, theses resources could be in the original site (cluster level) or, in another site (grid level). In both cases, `Saline` has just to redeploy snapshot images from the master to the new nodes.

Concerning the network, all the VMs are configured with two NICs: the *eth0* NIC exploits the NAT mode provided by the hypervisor and receives its MAC and IP addresses from the hypervisor. The *eth1* NIC exploits the bridge mode and is configured thanks to the *subnetID* and the *vmID* received from the master via the *eth0* NIC.

B. Experiments

In order to evaluate the cost of using `Saline`, we focus (i) on the overhead of the saving snapshot both with and without notifications and (ii) on the overhead of the snapshot deployment.

Experiments have been done on heterogeneous clusters of two Grid'5000 sites, Sophia and Nancy (from 1 to 64 nodes per site). The size of the exploited VM image is approximately 512 MBytes. Moreover, for a better understanding, we consider that only one VM could be executed on a physical node at each time.

1) Snapshot Saving: With or without notifications, the problem of snapshot management mainly concerns the copy of n VM images to one master which implies a network bottleneck at the master side. Indeed, without notifications, the framework has to save periodically the VM snapshots, and with notifications, the framework has to save the VM snapshots at a particular time. This section deals with (i) the overhead of the saving snapshot (both with and without notification callbacks) and (ii) the optimization proposed in Section III-A to efficiently retrieve the snapshot images.

[2]The Internet Assigned Numbers Authority: http://www.iana.org/

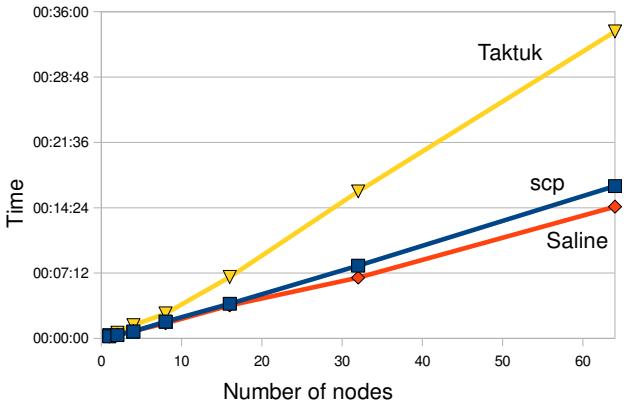

Figure 1. Snapshot Saving Analysis

Snapshot saving overhead (i). We compared three solutions: one copying with `scp`, another using `TakTuk`[3], and the last with our proposal implemented in `Saline`. Results visible in Figure 1 show that `Saline` has similar ($\leq$ 16 nodes) or better result than the two other tools to copy VM snapshots from several nodes to the master. Indeed, `Saline` is able to manage the network bandwidth of the master by limiting the number of packet collisions and lost packets.

Optimization in case of notification (ii). In addition, Figure 2 presents the percentage of node freed according to the time. The X-axis presents the time and the Y-axis presents the percentage of nodes freed. For this experiment, we used 64 nodes and measured how long it took to free each node. Results show clearly that, `Saline` is able to free 80% of all nodes in less than 2 minutes whereas other tools do the same after 17 minutes for `scp` and 29 minutes for `TakTuk`.

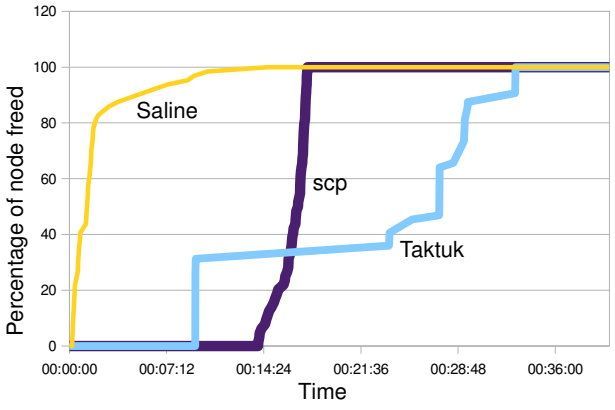

Figure 2. Detailed Snapshot Time Analysis

2) Snapshot Deployment: In this experiment we first deployed snapshot images at cluster level (Sophia site) and secondly at grid level (from Sophia to Nancy site). The following table presents the required time to deploy all VM snapshots in both cases. We currently exploit simultaneous `scp` commands to transfer VM snapshots from the repository node to the destination ones. Results show that the more physical nodes we have, the bigger the time to deploy all snapshots is. However, in the case of best-effort jobs, the time to deploy all VM snapshots ($\approx$ minutes) compared to the time to restart the best-effort job since the beginning ($\approx$ hours or days) is negligible. Moreover, results show that the time to restart snapshot images at cluster level is nearly the same as the time at grid level. This result could be explained by the network topology of G5K. Indeed, the network bandwidth between Sophia and Nancy is 10GB, whereas on a site, the network bandwidth between one node and its router is 1GB. In that condition, the number of nodes we use is negligible compared to the network bandwidth.

nb node	1	2	4	8	16	32	64
Loc. dep.	00:13	00:18	01:04	01:48	04:21	08:09	17:29
Dist. dep.	00:20	00:33	01:07	02:10	04:22	08:44	19:05

Time required to copy n VM snapshots from one master to n nodes locally (cluster level) and distantly (grid level). Time is given in min:sec.

V. Related Work

Historically, batch schedulers have been used to manage jobs on clusters. The combination of several jobs with several priorities on a limited number of resources is a well-known issue [15]. The back-filling model [16] is one of the most interesting solutions to this issue. To improve performance of back-filling, solutions like checkpointing were set up. These solutions are implemented in (i) user space, i.e., at application level (a linkage with special libraries is generally required) [17], [18] or (ii) in kernel space (generally using specific OS) [5], [6].

The use of VM snapshot capabilities could make application management easier. This is the reason why SGE [19] or Moab were upgraded to take advantage of VM functionalities. However these two projects are cluster-based and do not take into account grid constraints.

Concerning the Cloud area, some open-source projects like Nimbus [20], Eucalyptus [21] or SnowFlock [22] allow users to deploy their cloud and manage a virtual cluster on a physical one. However, to our knowledge, such systems don't deal with VM snapshot and/or migration at grid level.

Other projects focus on the deployment and management of virtual clusters by working on network virtualization (HIPCAL [23] or VIOLIN combined with VioCluster [24], [25]). However and once again, these works focus on a particular issue and do not address the VM management issue at grid level.

VI. Conclusion and Future Work

In this paper, we address the challenge of managing best-effort jobs by encapsulating them into VMs. We discussed two major issues that should be addressed at grid level: the management of VM images during the whole execution of the job and the network configuration and mobility of each VM involved into a job.

After discussing these points, we present our framework `Saline`, a generic and non-intrusive framework to deal with best-effort jobs in coordination with any RMS. Thanks to it, best-effort jobs can be launched into

[3]This tool is able to dynamically deploy the VM image from a central repository to the target nodes [14] – http://taktuk.gforge.inria.fr/

VMs and be transparently relocated from one cluster to another one each time it is required. `Saline` is able to benefit from RMS notification callbacks. In this case, `Saline` tries to migrate the VMs involved into the job. If no resources are available on another site, it makes the snapshot of the whole set of VMs, and waits until new resources become available. If no callbacks are present, `Saline` simply makes periodical snapshots of the set of VMs and checks their status. If VMs are not reachable *i.e.* resources have been withdrawn, `Saline` requests the RMS for new ones and restarts the job from the most recent snapshot. First experiments on our prototype show promising results with regard to current lost of computation in a grid such as Grid'5000.

Now, we are extending our framework by exploiting complementary mechanisms like monitoring probes. Indeed, which such probes, `Saline` will be able to optimize its scheduling decision (load-balancing between nodes/sites, prediction of hardware failure, ...).

More generally, we think `Saline` could contribute to make the use of scientific clouds easier.

ACKNOWLEDGMENT

The INRIA team carries out this research work in the framework of the XtreemOS project partially funded by the European Commission under contract #FP6-033576. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

REFERENCES

[1] J. Gallard *et al.*, "Saline: Improving Best-Effort Job Management in Grids," INRIA, Research Report RR-7055, 2009.

[2] R. Bolze *et al.*, "Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed," *International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 481–494, November 2006.

[3] G. Vallée *et al.*, "Tutorial: System-level Virtualization and Management using OSCAR," presented at the 5th Annual OSCAR Symposium (OSCAR 2007), May 15, 2007, Saskatoon, Saskatchewan, Canada. Co-hosted with HPCS 2007.

[4] B. Sotomayor *et al.*, "Combining batch execution and leasing using virtual machines," in *Proceedings of the 17th international symposium on High performance distributed computing (HPDC)*. New York, NY, USA: ACM, 2008.

[5] P. H. Hargrove *et al.*, "Berkeley lab checkpoint/restart (BLCR) for Linux clusters," *Journal of Physics: Conference Series*, vol. 46, pp. 494–499, 2006.

[6] D. Thain *et al.*, "Distributed Computing in Practice: the Condor Experience," *Concurr. Comput. : Pract. Exper.*, 2005.

[7] A. Iosup *et al.*, "The Grid Workloads Archive," *Future Gener. Comput. Syst.*, vol. 24, no. 7, pp. 672–686, 2008.

[8] F. Hermenier *et al.*, "Entropy: a Consolidation Manager for Clusters," in *Proceedings of the International Conference on Virtual Execution Environments (VEE)*, 2009.

[9] M. Stillwell *et al.*, "Resource allocation using virtual clusters," in *the 2009 9th Inter. Symp. on Cluster Computing and the Grid (CCGRID)*, Washington, DC, USA, 2009.

[10] J. Gallard *et al.*, "VMdeploy, Improving Best-Effort Job Management in Grid5000," INRIA, Research Report RR-6764, December 2008.

[11] W. Emeneker *et al.*, "Increasing Reliability through Dynamic Virtual Clustering," in *High Availability and Performance Computing Workshop*, 2006.

[12] H. K. F. Bjerke *et al.*, "Tools and techniques for managing virtual machine images," in *Proceedings of the 3rd Workshop on Virtualization in High-Performance Cluster and Grid Computing (VHPC)*, 2008.

[13] N. Capit *et al.*, "A batch scheduler with high level components," in *Proceedings of the 5th IEEE Intern. Symposium on Cluster Computing and the Grid (CCGrid)*, 2005.

[14] B. Claudel *et al.*, "TakTuk, adaptive deployment of remote executions," in *Proceedings of the 18th ACM international symposium on High performance distributed computing (HPDC)*, New York, NY, USA, 2009, pp. 91–100.

[15] M. W. Margo *et al.*, "Impact of Reservations on Production Job Scheduling." Workshop on Job Scheduling Strategies for Parallel Processing, 2007.

[16] E. Shmueli *et al.*, "Backfilling with Lookahead to Optimize the Performance of Parallel Job Scheduling," in *Job Scheduling Strategies for Parallel Processing*, 2003.

[17] J. S. Plank *et al.*, "Libckpt: Transparent Checkpointing Under Unix," in *Proceedings of the USENIX 1995 Technical Conference (TCON)*, Berkeley, CA, USA, 1995.

[18] J. F. Ruscio *et al.*, "DejaVu: Transparent User-Level Checkpointing, Migration and Recovery for Distributed Systems," in *Supercomputing (SC)*, New York, NY, USA, 2006.

[19] N. Fallenbeck *et al.*, "Xen and the art of cluster scheduling," in *Proceedings of the 2nd Int. Workshop on Virtualization Technology in Distributed Computing (VTDC)*, 2006.

[20] K. Keahey *et al.*, "Contextualization: Providing One-Click Virtual Clusters," in *Inter. Conference on e-Science*, 2008.

[21] D. Nurmi *et al.*, "The Eucalyptus Open-source Cloud-computing System," in *the 9th Inter. Symposium on Cluster Computing and the Grid (CCGRID)*, Shanghai, China, 2009.

[22] H. A. Lagar-Cavilla *et al.*, "SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing," in *the 3rd European Conf. on Computer Systems (Eurosys)*, Germany, 2009.

[23] P. Primet/Vicat-Blanc *et al.*, "HIPCAL: State of the Art of OS and Network virtualization solutions for Grids." INRIA: Research Report, September 2007.

[24] X. Jiang *et al.*, "Violin: Virtual Internetworking on Overlay Infrastructure," in *Parallel and Distributed Processing and Applications*. Tech. Rep. CSD TR 03-027, Department of Computer Sciences, Purdue University, 2003, pp. 937–946.

[25] P. Ruth *et al.*, "VioCluster: Virtualization for Dynamic Computational Domains," in *CLUSTER*, 2005.