# DEVA: Distributed Ensembles of Virtual Appliances in the Cloud

David Villegas and S. Masoud Sadjadi

School of Computing and Information Sciences
Florida International University
Miami, FL, USA
{dvill013, sadjadi}@cis.fiu.edu

**Abstract.** Low upfront costs, rapid deployment of infrastructure and flexible management of resources has resulted in the quick adoption of cloud computing. Nowadays, different types of applications in areas such as enterprise web, virtual labs and high-performance computing are already being deployed in private and public clouds. However, one of the remaining challenges is how to allow users to specify Quality of Service (QoS) requirements for composite groups of virtual machines and enforce them effectively across the deployed resources. In this paper, we propose an Infrastructure as a Service resource manager capable of allocating *Distributed Ensembles of Virtual Appliances* (DEVAs) in the Cloud. DEVAs are groups of virtual machines and their network connectivities instantiated on heterogeneous shared resources with QoS specifications for individual entities as well as their connections. We discuss the different stages in their lifecycle: declaration, scheduling, provisioning and dynamic management, and show how this approach can be used to maintain QoS for complex deployments of virtual resources.

## 1 Introduction

Infrastructure as a Service (IaaS) clouds allow users to instantiate Virtual Machines (VMs) on demand in remote shared resources for a certain period of time. One of the currently faced challenges in such systems is allowing users to specify fine-grained requirements for groups of resources and ensure that the promised Quality of Service (QoS) is met for them, not only in terms of individual machines, but also in their aggregate traffic assignment. This requirement is essential to run certain parallel and distributed workloads such as scientific applications that rely on low network latencies or high bandwidth, for example. We propose an IaaS cloud manager to tackle this problem at different levels: user request definition, scheduling of virtual resources and management of physical infrastructure to secure the requested service.

In order to maximize resource utilization, providers assign VMs to shared physical infrastructure. Consequently, mechanisms need to be implemented to ensure that utilization is fairly distributed according to the requested allocation. These measures have to consider various aspects such as VM placement,

creation of virtual network links between them that provide the appropriate bandwidth and latency, and dynamic monitoring and management of the composite allocations. Our proposed work in this paper is an attempt to address the above mentioned issues in the current IaaS implementations such as Amazon [1], OpenNebula [14], Eucalyptus [11] or Nimbus [6].

Our approach allows users to submit requests by specifying the requirements of their application. A cloud resource manager is in charge of brokering for the appropriate resources and acquiring them for the desired time. This process is akin to the act of planning for traditional computing equipment, where hardware —architecture, processor speed, memory, switching and routing devices, or machine interconnections— can be carefully tailored based on costs and capabilities, except with the benefits of the cloud, such as elasticity or pay-per-use. An additional advantage is that, by defining concrete Service Level Agreements (SLA), the broker can use heterogeneous resources, reducing the fragmentation between clouds with different capabilities. This fact can also be employed to enable federation among providers.

In this paper, we implement a heuristic placement algorithm based on the *assign* mapping method used in Emulab [4] for network topologies. Next, we create a cloud interface on top of OpenNebula which accepts user requests for Distributed Ensembles of Virtual Appliances (DEVAs) with annotated network connections and VM descriptions. We also define *DEVA agents* in charge of enforcing QoS, isolating traffic, monitoring resource usage and tunneling packets between remote resources to seamlessly create layer 2 networks among ensemble members. Finally, we perform experiments to validate our architecture and demonstrate how QoS can be fulfilled.

The results demonstrate that our system can be used to instantiate groups of VMs in clouds with user-defined QoS requirements to execute different types of applications. The DEVA IaaS manager enables fine-grained control over the allocated resources, allowing users to request the appropriate infrastructure and providers to apply the right policies so that resources are not allocated in excess.

## 2 System Overview

Figure 1 depicts the general architecture of the system and the flow of interactions among the different components. Users prepare a request description based on their application requirements and send it to the Cloud Interface component. When a request is received, it is parsed and a graph describing the virtual deployment is generated. The Mapper component tries to find the most appropriate resources based on pre-defined site policies and the fulfillment of the user specified requirements. The resulted mapping is then sent to a VM provisioner, for example, the OpenNebula [14], which is the one that we used in our prototype implementation. The provisioner transfers the required VM images to the destination nodes, instantiates them, and notifies the DEVA Agents in every hosting node. Each agent then applies the appropriate network mechanisms to perform traffic monitoring, isolation and traffic control. Agents also monitor the state of

the VMs and notify the central manager. The steps mentioned in the process are enumerated and discussed in detail next.

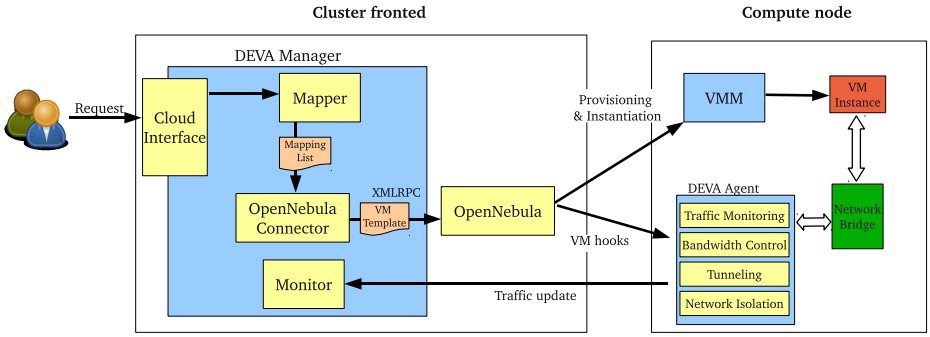


**Fig. 1.** General architecture

We define a *Distributed Ensemble of Virtual Appliances* (or DEVA) as a group of virtual appliances, including virtual machines, virtual network devices, and their connections, altogether with a set of QoS requirements applied either globally or to individual members of the ensemble. DEVAs can be described using XML, and sent to a resource manager capable of processing and instantiating them within a pool of physical resources. Our current implementation supports single VM requirements such as CPU power and memory, network bandwidth in megabits per second and latency, as *low* or *high*.

## 3 DEVA Manager

### 3.1 Mapping of DEVAs

After a DEVA description is sumitted to the DEVA Manager, the Mapper module decides where to place each of the individual components. During this process, there is a match-making algorithm that selects those resources that can fulfill the request: this stage considers both individual VMs (*i.e.,* available CPU and memory), and the whole ensemble (network connectivity, available bandwidth, *etc.*). We assume that physical resources may be heterogeneous, and that they may belong to different administrative domains. We implement a centralized match-making approach, where the main process is on the DEVA manager front-end node and has all the information about the available resources, even if they are located on different sites.

As it has already been discussed in the literature, the process of mapping a virtual topology to a physical one is an NP-hard problem, making comprehensive algorithms too costly. Instead, we have adopted the *assign* algorithm [12], used in emulab [4], for the mapping stage.

Our implementation of *assign* differs from the original one in various aspects based on the different targeted use. While this algorithm was originally designed to solve the so called *network testbed mapping problem* (*i.e.*, how to find an

optimal or close to optimal mapping from a virtual network topology to a phisical one), we take a more pragmatic approach, focusing on providing the required connectivity and QoS rather than an exact replica of the topology.

The algorithm considers three types of connection mapping with different scores that lead to various solutions. These types are:

- **Trivial**: Both VMs share the same physical host, thus sharing an internal connection.
- **Direct**: The hosting machines are directly connected through a cable.
- **IntraSwitch**: The hosting machines are connected to the same switch.
- **InterSwitch**: The hosting machines are connected to different switches which in turn are connected through a cable.
- **InterRouter**: The hosting machines are in different layer 2 networks, connected by one or more routers in between.

Each of the itemized connection types has a cost in terms of latency and possible bandwdith, which is accounted for in the mapping process. The algorithm gives each connection a score, promoting results with better connectivity. The case of *InterRouter* connections is special since, when a connection between two machines that share a logical layer 2 link is mapped to this kind of connection, tunneling will be necessary at the provisioning stage, which also results in additional network latency. The algorithm takes this fact in consideration, creating a policy violation if the mapped connection won't be able to fulfill the request. In particular, low latency links from the DEVA request mapped to InterRouter connection may result in a policy violation.

### 3.2 DEVAs across heterogeneous resources

The previous phase of the process is in charge of finding a good mapping between the virtual topology and the physical resources. The *assign* algorithm outputs a list of pairs of *virtual resource* to *physical resources* mappings altogether with the policies in the links. The next step takes this mapping and realizes it.

We use the *OpenNebula* virtual infrastructure manager, version 1.4, to provision and control individual machines. *OpenNebula* is an IaaS resource manager that receives requests via a command line interface or remote procedure calls and instantiates the appropriate VMs, described by a template file. The DEVA manager receives the output of the infrastructure mapper and translates it to calls to *OpenNebula*'s XMLRPC protocol in order to provision and start the VMs. The process to realize a DEVA involves four steps:

1. Translate the original DEVA request into *OpenNebula* VM templates
2. Send instantiation requests to *OpenNebula*, using the mapping results to indicate the appropriate hosting machines
3. Apply the network configuration at each machine to ensure QoS is met
4. Monitor traffic at each host and aggregate it at the DEVA manager to form a picture of the overall state

*OpenNebula's* VM template requires some information that is provided in the original DEVA request, such as the location of the kernel, ramdisk and filesystem images. Other data is generated by the DEVA manager, such as the VM's MAC address, and the rest of information is provided by the mapping algorithm, for example the destination host. The template is created dynamically and sent to *OpenNebula*, which is in charge of transferring the required images from a central repository to the host machine and starting the VM. Each of the instantiated VMs is identified by its uniquely generated MAC address, controlled by the central DEVA manager.

The next step consists of applying the required network configuration at each host in order to perform traffic monitoring, network isolation, bandwidth management and intelligent routing between *ensemble* members. To accomplish this, each physical machine needs to know the details about the hosted VM's network configuration. We use *OpenNebula's* hook functionality for this. Hooks are small scripts that can be configured to run at certain points of *OpenNebula's* request lifecycle, such as when a VM is started, stopped or removed. They are executed either in the head node or in the hosting machine. Since we need a process to manage network settings accordingly to the original user's request at each node, we have developed a daemon (called a *DEVA Agent*) that runs at each host that can perform these actions.

## 4 DEVA Agents

A *DEVA agent* runs as a background process that listens for new requests, runs some pre-defined commands on the host machines, monitors network and VM behavior, and creates VPN (Virtual Private Network) tunnels between sites. The DEVA Manager notifies the appropriate agent of a new VM member using a hook, which sends a command through the agent's specified port. There is one designated agent per site that creates VPN tunnels, called a *site gateway*. All agents in a site know the address of the site gateway and can send requests to create new tunnels. There are three supported commands: ADD, DEL and TUN.

When an agent receives an ADD request, it looks for the virtual network interface of the specified VM and queries the DEVA manager to retrieve global information about the DEVA, such as which ensemble members this VM is connected to and what is the requested network QoS. After this information is returned, the agent can perform the appropriate actions. In the case that one or more of the ensemble members share a virtual layer 2 connection with VMs assigned to machines in different domains, the agent on the host machine of the newly assigned VM issues an additional call to its corresponding site gateway asking for a tunnel to be created between the VMs using the TUN command. Finally, the DEL command is analogous to ADD, which basically removes a VM from a host.

Different DEVAs are completely isolated from one another at network layer 2. When the DEVA agent receives a request to add a new VM, it creates *ebtables* [1] rules to block all traffic except those frames originating from or directed to other ensemble members in the same logical network.

When an ADD request is received by the agent, it retrieves a list of the VM's neighbors from the DEVA manager, and then it adds two rules for each of them: one to allow outgoing traffic from the VM's unique MAC address to the neighbor's one, and the reciprocal rule to accept packets originating from the neighbor's address with the local VM as the destination. ARP requests are special since they do not have a unique target, and therefore an additional rule is added to allow this kind of packets from and to the network.

In the original DEVA request, each link in the ensemble may be annotated with a desired latency and bandwidth. The *DEVA Agent* consults the manager to retrieve the bandwidth constraints between the local VM and each of its neighbors. For each pair, the agent creates a queuing class discipline in the kernel's traffic control module using the *tc* command. Next, packets are marked in the kernel and filtered to use the appropriate class. We use the Hierarchical Tocken Bucket (HTB) for its versatility and good performance.

*Site gateways* are in charge of routing frames that are not targetted to the host's Local Area Network. Since DEVAs may be distributed among different networks that are not reachable at layer 2, agents must encapsulate the frames that are directed to another network and send them. In our architecture, each site must have a VPN server and allow client connections from other sites. Also, each site that needs to tunnel traffic has to have at least one site gateway. When the site gateway agent starts, it runs a VPN client for each of the remote sites and connects to them. The client is configured to create a special *tap* device, in such a way that all traffic send through a tap will be tunneled to a different site. Then agents that instantiate new VMs retrieve the neighboring VM hosts, and for each host that is located outside of the VM's domain, they send a TUN request to the local site gateway so that packets originating from the VM directed to the destination neighbor are tunneled through the appropriate channel.

## 5 Experimental Results

Here we perform different experiments to demonstrate the use of DEVAs as a viable cloud resource management approach. We have run several measurements to quantify the performance and scalability of our design and the prototype implementation. We evaluate the following hypothesis experimentally:

1. The overhead of the DEVA agents is small when executing High-Performance Computing applications.
2. Traffic is effectively isolated between different deployed DEVAs.
3. User requested QoS is fulfilled through the execution of applications.

[1] http://ebtables.sourceforge.net

For all experiments, we employ our Magellan cluster, which is composed of 8 nodes, each of them with a Pentium 4 CPU at 3 GHz and Hyper Threading and 1 GB of memory. The nodes are connected using a 1 Gbps ethernet link and a Gigabit switch. The head node of the cluster has two network interfaces, one is connected to the Internet and the other to the private network where the other 7 nodes are also connected. Each node runs CentOS 5.3 and the Rocks cluster administration software version 5.2 with the Xen roll, which provides Xen 3.0.3.

### 5.1 Overhead measurement

In the first set of experiments, we quantify the overhead imposed by the agents at each host. The agents control network isolation and bandwidth usage. Each agent manages incoming and outgoing traffic at the virtual network interface of each VM by filtering packets based on the source and destination addresses. In our prototype implementation, since each VM has a unique MAC address, the agent can control traffic for each pair of ensemble members, which implies that there are two ebtable rules for each pair to manage incoming and outgoing traffic.

For this experiment, we execute the Weather Research and Forecast (WRF) package [8], a simulation software used for atmospheric research, using different setups. WRF uses the Message Passing Interface (MPI) middleware [2] for communication between processes, and can be executed with different number of processes. We have previously studied the behavior of this program [7], and demonstrated that its communication model is highly sensitive to network delays. In particular, link latencies of more than 1 ms result in slower executions when adding more nodes to the computation, making it impossible to perform blind scaling across Internet.

In this macro-benchmark, we perform executions of WRF for 1, 2, 4 and 8 nodes using the physical cluster running Xen's Domain 0, a set of OpenNebula instantiated VMs, and a DEVA instantiated through our manager. Each process is assigned to a different node. For all runs that involve VMs, we use a base CentOS 5.3 Xen image with 512 MB of memory and a 2.6.18 Linux kernel. We installed the required software to compile and run WRF version 2.2.

Figure 2 shows the execution time for the three considered cases, namely the physical cluster, the VMs instantiated through OpenNebula and the DEVA VMs. The slowdown between the physical execution and OpenNebula's execution is entirely produced by the Xen virtualization overhead. It can be observed that when the number of nodes grows, so does the amount of communication among them. This factor is specially significant, since I/O virtualization is known to have a huge performance hit. The use of more sophisticated network drivers and newer hardware would certainly alleviate the slowdown, although this is outside the scope of the current paper. The overhead produced by the filtering and monitoring by the agents is minimal in this case, averaging to less than 1 percent.

[2] http://www.mpi-forum.org

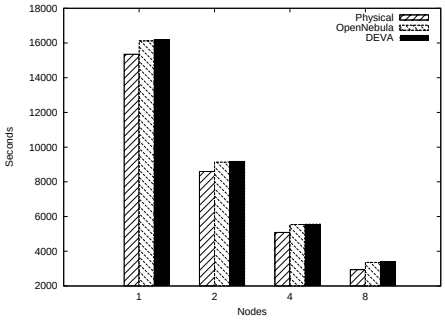


**Fig. 2.** WRF Execution time

Overhead micro-benchmarks: The next experiment performs a micro-benchmark of the same parameters as the previous one. We investigate the performance impact of adding the necessary filtering rules to provide isolation between VMs in different networks. For this experiment, we instantiate two VMs connected to a virtual switch, but we add rules as if there were a greater number of VMs in the DEVA. Since rules are directly dependent on the number of connections a VM has to other VMs, this allows us to measure the slowdown produced by those rules in relation to the size of the broadcast group.

Figure 3 shows bandwidth, round trip time and number of generated rules in relation to the VM connections. We use *netperf* to measure the total available bandwidth between two physical machines, two VMs instantiated with 0, 10 and 100 connections instantiated through the DEVA manager. Note that the number of rules depends on how many ensemble members are in the same broadcast group (*i.e.*, share the same virtual link) and not the size of the DEVA.

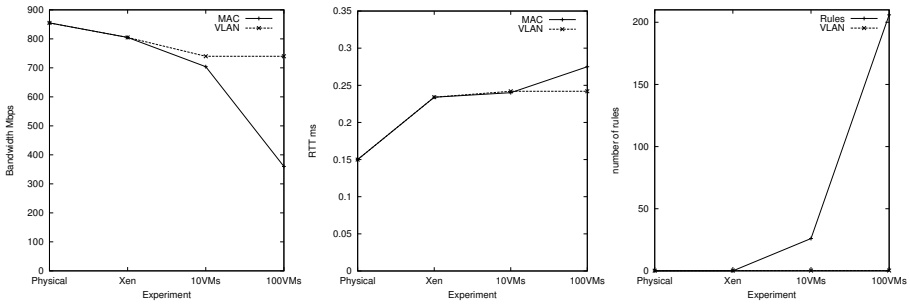


**Fig. 3.** Overhead introduced by filtering rules

From the results, we can see that the impact of adding more VMs is increasingly high, specially in terms of total bandwidth. Although 10 connections still has a tolerable overhead of 6.2% over the zero-connection case, adding more

VMs results in a great overhead due to the number of filtering rules that need to be added.

The approach of assigning unique MAC addresses to control network isolation is therefore not scalable for large groups of ensemble members. To address this problem, we employ VLANs to identify which virtual network each packet belongs to. VLANs are defined by adding 4 additional bytes to each datagram at the source interface. To implement this method, we modified the DEVA manager to assign a unique id to each broadcast group. The downside of this solution is that VLANs need to be supported by the physical switch, while the individual MAC filtering is network agnostic.

### 5.2 Isolation and QoS conservation

Isolation is also demonstrated in the provided QoS. Traffic from one DEVA should not impact bandwidth and latency allocated by the manager to another one. The only exception to this is when best effort links are requested, in which case no guarantee is made by the system.

In the next experiment, we test the effects of multiple DEVAs running in our Magellan cluster. First we execute two applications that share the same physical resources, and we constrain the allocated bandwidth to ensure each of them has the requested QoS. We compare it to the same case when allocating the VMs individually through the IaaS manager, *OpenNebula*.

For this experiment, we choose two applications with different behaviors: first, we create HTTP traffic between two VMs, one of them with the Apache Web server version 2.2.3, and the other with the *apache benchmark* tool. Next, we simulate network traffic by transferring files between another pair of VMs. The two clients share one host, and the two servers are placed in another host.

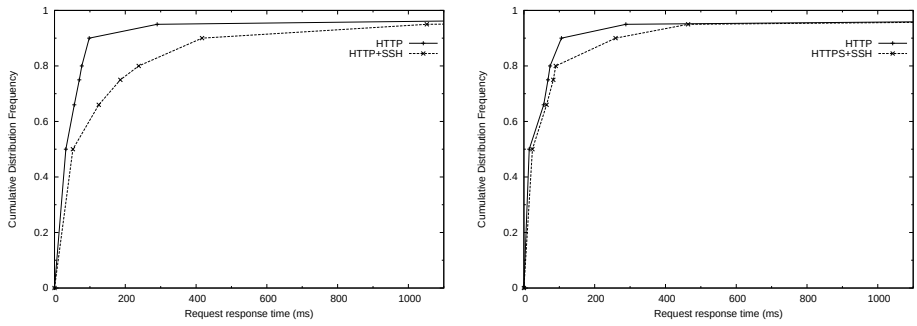


**Fig. 4.** HTTP traffic between two VMs with and without network contention. Right figure uses DEVA traffic control mechanisms to fulfill requested QoS

Figure 5.2 shows the cumulative distribution function of the Web server response time. The left figure shows unmanaged network traffic: while half of the requests take similar time, the waiting time for the other half increases up to four times with the additional network load. In the right figure, two DEVAs are

requested with different requirements: the pair of VMs with the HTTP traffic has a 600 Mbps virtual link, while the other two VMs have a virtual link of 40Mbps. It can be seen that the response time when additional traffic is generated remains similar to the case in which only the HTTP requests take place. As the figure shows, 80% of requests are completed in the same time, while the top 20% experience delays up to three times.

### 5.3 Use of heterogeneous DEVAs and resources

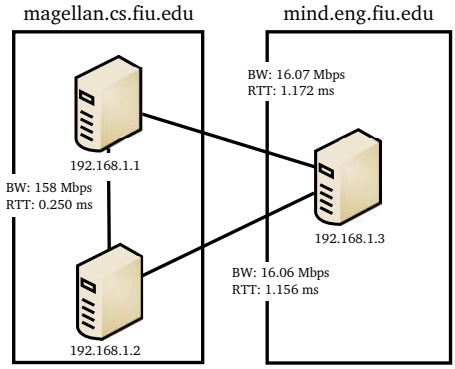


**Fig. 5.** Intersite deployment of a DEVA

Finally, we run an experiment to demonstrate how VMs in a DEVA can be placed across different administrative domains while QoS is enforced. In this case, we provision a DEVA with three VMs: two of them are connected among them with a 160 Mbps virtual link, and the third is connected to the first two with a 16 Mbps link. Next, we reduce the available nodes in the Magellan cluster to two and add a physical machine from another cluster, Mind, located in a different campus at FIU. The *assign* algorithm maps two of the VMs to Magellan and the third one to Mind, and creates a VPN tunnel among them to provide a virtual network. We calculate link bandwidth among VMs by using *netperf* and the round trip time by averaging 50 pings between the machines. Figure 5 shows how the virtual ensemble maintains the requested QoS.

## 6 Related Work

Amazon EC2 [1] is perhaps the most prominent example of IaaS public cloud. Users can request any number of virtual machines to be instantiated in the shared infrastructure. VM capabilities are defined by the requested instance type, and price is set accordingly to the time and characteristics of the used VMs. One of the main shortcomings of EC2 is the lack of QoS assurances for network traffic:

while processor, memory and disk capabilities are well defined, users can't make assumptions about the network.

Eucalyptus [11], OpenNebula [14] and Nimbus [6] are IaaS cloud implementations that have some similarities with EC2. However, these solutions do not support composite groups of VMs with a defined network QoS in the requests. Another difference is in the deployment of VMs accross different domains. *OpenNebula* supports interoperation by implementing different protocols such as the Open Cloud Computing Interface (OCCI) or by extending the local resources into public clouds. In our case, we the DEVA manager decides when to create a tunnel to connect VMs in different sites based on the requested QoS.

Another type of solutions focus on the network virtualization aspect, rather than in the resource management and providing an interface for users to manage composite groups of virtual resources. VIOLIN [13], VNET [15], VINE [16] or IPOP [3] are examples of such systems.

Also, DEVAs have similitudes with virtual clusters such as [9] or [10]. Differently than in our work, these solutions focus on instantiating the required virtual resources and providing the appropriate software and network configuration.

Finally, our work has points in common with network testbeds, where many of the problems of provisioning execution environments to replicate network topologies have to be solved. Emulab [4] allows users to create network experiments over shared resources by requesting a configuration of virtual hosts and connections. The main difference from our work and Emulab is that the latter is principally targeted for repeatable network experiments, while our system is designed to host virtual environments to run possibly long lasting applications. Also, since our primary goal is not to replicate the user's network characteristics, we can make some optimizations in the requested topologies. In GENI, [2] describes a similar approach in which ORCA [5] is extended to support additional networking infrastructure to create multi-site VM deployments via VLAN tags. Our work is more focused in the placement aspect and QoS fulfillment.

## 7 Conclusions and Future Work

We have described an approach to instantiate groups of Virtual Machines in the cloud while fulfilling their composite QoS requirements. Our experiments indicate that this implementation is viable and can be used to execute different workloads with specific network and processing requirements. As future work, we plan to further investigate the dynamic behavior of DEVAs, and how to respond to varying traffic and resource utilization. The DEVA manager can perform actions to further control QoS of the sytem by migrating VMs among resources, or adjusting the allocations according to the global state. Finally, we want to explore different placement policies among sites to accomplish site-specific and global goals such as lower power utilization or higher throughput.

## References

1. Amazon elastic compute cloud. http://aws.amazon.com/ec2.

2. I. Baldine, Y. Xin, A. Mandal, C. H. Renci, U.-C. J. Chase, V. Marupadi, A. Yumerefendi, and D. Irwin. Networked cloud orchestration: A geni perspective. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 573 –578, 2010.
3. A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo. Ip over p2p: Enabling self-configuring virtual ip networks for grid computing. In *In Proc. of 20th International Parallel and Distributed Processing Symposium (IPDPS-2006)*, pages 1–10, 2006.
4. M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. Large-scale virtualization in the emulab network testbed. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 113–128, Berkeley, CA, USA, 2008. USENIX Association.
5. D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. G. Yocum. Sharing networked resources with brokered leases. In *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pages 18–18, Berkeley, CA, USA, 2006. USENIX Association.
6. K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron. Virtual workspaces in the grid. In J. C. Cunha and P. D. Medeiros, editors, *Euro-Par 2005 Parallel Processing*, volume 3648 of *Lecture Notes in Computer Science*, pages 421–431. Springer Berlin / Heidelberg, 2005.
7. J. C. Martinez, L. Wang, M. Zhao, and S. M. Sadjadi. Experimental study of large-scale computing on virtualized resources. In *Proceedings of the 3rd International Workshop on Virtualization Technologies in Distributed Computing (VTDC 2009) of the IEEE/ACM 6th International Conference on Autonomic Computing and Communications (ICAC-2009)*, pages 35–41, Barcelona, Spain, June 2009.
8. J. Michalakes, J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang. Reseach and Forecast Model: Software Architecture and Performance. In *11th ECMWF Workshop on the Use of High Performance Computing In Meteorology*, pages 156–168, Reading, UK, October 2004.
9. M. A. Murphy, B. Kagey, M. Fenn, and S. Goasguen. Dynamic provisioning of virtual organization clusters. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 364–371, Washington, DC, USA, 2009. IEEE Computer Society.
10. H. Nishimura, N. Maruyama, and S. Matsuoka. Virtual clusters on the fly - fast, scalable, and flexible installation. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:549–556, 2007.
11. D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:124–131, 2009.
12. R. Ricci, C. Alfeld, and J. Lepreau. A solver for the network testbed mapping problem. *SIGCOMM Comput. Commun. Rev.*, 33:65–81, April 2003.
13. P. Ruth, X. Jiang, D. Xu, and S. Goasguen. Virtual distributed environments in a shared infrastructure. *Computer*, 38:63–69, May 2005.
14. B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13:14–22, 2009.
15. A. I. Sundararaj and P. A. Dinda. Towards virtual networks for virtual machine grid computing. In *Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium - Volume 3*, pages 14–14, Berkeley, CA, USA, 2004. USENIX Association.
16. M. Tsugawa and J. Fortes. A virtual network (vine) architecture for grid computing. *International Parallel and Distributed Processing Symposium*, 0:123, 2006.