Elastic Management of Tasks in Virtualized Environments

Íñigo Goiri, Jordi Guitart and Jordi Torres [1]

Abstract— **Nowadays, service providers in the Cloud offer complex services ready to be used as it was a commodity like water or electricity to their customers. A key technology for this approach is virtualization which facilitates provider's management and provides on-demand virtual environments, which are isolated and consolidated in order to achieve a better utilization of the provider's resources.**

However, dealing with some virtualization capabilities, such as the creation of virtual environments, implies an effort for the user in order to take benefit from them. In order to avoid this problem, we are contributing the research community with the EMOTIVE (Elastic Management of Tasks in Virtualized Environments) middleware, which allows executing tasks and providing virtualized environments to the users without any extra effort in an efficient way.

This is a virtualized environment manager which aims to provide virtual machines that fulfils with the user requirements in terms of software and system capabilities. Furthermore, it supports fine-grained local resource management and provides facilities for developing scheduling policies such as migration and checkpointing.

Keywords— **Cloud Computing, Virtual Machines, Resource Management**

I. Introduction

Cloud Computing is becoming more and more important everyday style of computing where computation is provided over the Internet rather than being locally on the user's machine. Locating these services on third-party companies makes the users able to access them anytime and anywhere. Thanks to the Cloud, the service provider can virtualize its resources and dynamically provision them as unified computing resources.

In order to be profitable, service providers tend to share their resources among multiple concurrent services owned by different customers, which implies the cohabitation of services with very different behaviors and requirements. In order to solve this challenge, virtualization has become a key technology in the Cloud. It allows the consolidation of applications, multiplexing them onto physical resources while supporting isolation from other applications sharing the same physical resource. In addition, virtualization has other valuable features for service providers like the image of a dedicated and customized machine to each user, decoupling them from the system software of the underlying. Moreover, it also offers new capabilities such as migration or checkpointing and fine-grained resource management.

Nevertheless, dealing with some virtualization capabilities, such as the creation, implies an effort for the user in order to take benefit from them. In order to avoid this problem, we are contributing the research community with the EMOTIVE (Elastic Management of Tasks in Virtualized Environments) middleware, which allows executing tasks and provides virtualized environments to the users without any extra effort in an efficient way.

This is a virtualized environment manager which aims to provide virtual machines that fulfils with the user requirements in terms of software and system capabilities while exploiting the features of virtualization in a new approach for facilitating providers' management. In addition, it provides resource management at several layers, namely locally to each node in the provider and among different nodes in the provider,

Next sections present the core of this middleware in a detailed way and describe its capabilities in detail. Finally, we evaluate it and demonstrate its potential for being used in current Cloud providers.

II. Architecture

In order to perform resource management in Cloud providers, we propose the architecture shown in Figure 1. This architecture considers three different layers: *Scheduler*, *Resource Management*, and *Resource Fabrics*. The *Scheduler* layer comprises all the global resource management decisions, both among different providers and different nodes in a single provider. This layer is in charge of deciding where a task will be executed and managing its location during the execution (e.g. migrations, cancellations, etc.). The *Resource Management* layer comprises all the local resource management decisions (i.e. in a single node). This layer is in charge of managing the physical resources in a node using virtualization and distributing them among the VMs running on that node. Finally, the *Resource Fabrics* layer comprises the physical resources where the VM will run.

In this article, we describe the implementation of the aforementioned architecture which is part of the core of EMOTIVE Cloud [1]. Firstly, the *Scheduler* layer supports several schedulers with different features, depending on which resource allocation policies want to be carried out. The responsibilities of this layer are further described in Section V.

The *Resource Management* layer is implemented by means of *Virtualized Resource Management and Monitoring (VRMM)*. This layer is in charge of distributing the resources in a single node among the services running on it using virtualization capabilities. In addition, it continuously monitors the resource usage of these services and the fulfillment of

[1]Universitat Politècnica de Catalunya (UPC), Barcelona Supercomputing Center (BSC) e-mail: {igoiri,jguitart,torres}@ac.upc.edu.

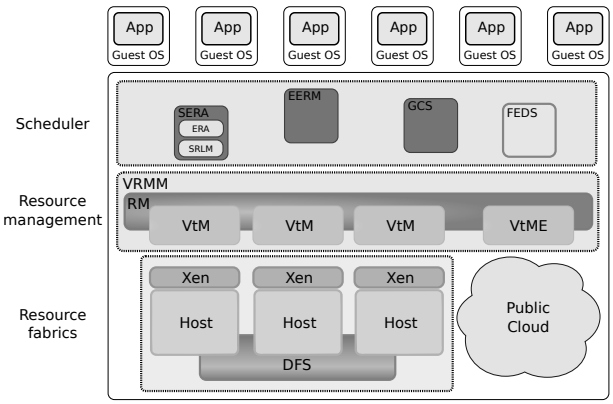

Fig. 1
EMOTIVE Cloud Architecture

their SLAs. If any SLA violation is detected, an adaptation process for requesting more resources to the provider is started, first locally in each node, then globally in the provider, and finally with other providers. The responsibilities of the *Resource Management* layer are further described in Section IV.

Finally, the *Resource Fabrics* layer virtualizes the resources by means of the Xen Hypervisor [2]. In addition, it also implements a distributed file system (DFS) that supports efficient VM creation, migration and checkpointing. In addition, the file system includes a global repository where clients can upload the input files needed by the tasks to be executed in the provider (i.e. stage-in phase) and retrieve the output files (i.e. stage-out phase).

The three layers will be described in a detailed way in next sections from the basement to the top in order to give a incremental view of the overall architecture.

III. Resource fabrics

The basements of the system are the resources which allow running Virtual Machines on top of them. In order to fulfill all the requirements of the system, it requires different facilities such as storage and networking. In addition, these facilities must make the system able to support different capabilities such as migration or checkpointing that will be used by upper layers.

Current implementation supports virtualization by using the Xen hypervisor [2]. In order to work with this hypervisor and manage low-level issues, we have developed a software called XenMonitor. It is in charge of abstracting the Xen layer about low-level issues. It does not only need obtaining information of the VMs but it also requires modifying some low-level capabilities of the hypervisor such as the CPU policies.

This software makes use of the XenAPI [3] which access to the Xen remote facilities and the XenStat library which directly access to the Xen hypervisor in order to obtain different metrics like the CPU used by a VM or the disk accesses it is doing. Moreover, it makes uses of XenStore for stablishing a bridge for communicating VMs with the management domain what makes the system able to obtain information like the IP of the VM or the memory it is actually using. Although other alternatives like libvirt [4] exists, they are intended for a generic use with other virtualization approaches and they do not support low-level issues related with Xen. Future work plans to create a merge between libvirt and XenMonitor, which will make the programmer able to access to other hypervisors while accessing to concrete stuff of the Xen hypervisor.

Data management is another big challenge in order to implement a Cloud provider and the first issue it must deal with is supporting an efficient access to VMs disks while enabling migration. It is done by making each node access its own local disk and the disk of the other nodes (using NFS). This allows each node creating VMs and executing tasks efficiently using the local disk. Furthermore, tasks can be also migrated with minimum overhead, since it is not necessary to transfer the VM image, while maintaining their accessibility during the whole process. Moreover, migrated tasks can access remotely their required data without noticeable performance penalty. Further details on these capabilities can be found in [5].

As the architecture supports fault-tolerance by adding VM checkpointing, it must be able to store these checkpoints taking into account two main factors. On one side, the checkpoint mechanism is fault-tolerant and any single point of failure must be eliminated. On the other side, checkpoint can be concurrently recovered from different nodes in order to resume task execution faster after a node crash under contention. Both requirements are achieved by a checkpoint storage implemented using a Hadoop Distributed File System [6] which distributes and replicates the checkpoints in all the nodes of the provider.

A way for accessing data and store it is required for persistently store users' data. The system supports different methods for storing data such as FTP, SFTP, S3 and whatever storage system that Hadoop FileSystem supports. Thanks to this support, it allows submitting a task to a VM with some data associated or storing output files. In addition, it can also store the whole VM image in order to retake its execution in a future.

Finally, the *Resource Fabrics* layer also supports an addressing and naming networking system, which provides access to the VMs. It uses a DHCP server that dynamically assigns an IP address to each VM and updates the local DNS server in order to access in a human-friendly way to them. Thanks to this naming, the system can access to the VM using a SSH wrapper, that allows submitting tasks using JSDL, which describes the tasks and allows attaching input data (using the data managemet support).

IV. Resource management

The main responsibility of the *Resource Management* layer is to manage VMs and distribute the re-

sources in a single node among the VMs running on it. This is carried out by *Virtualized Resource Management and Monitoring (VRMM)*, which can be seen as a wrapper of the physical machine that allows creating new customized VMs, executing tasks on these VMs, monitoring their execution, extracting the results, and destroying the VMs. A single *Virtualization Manager (VtM)* per physical node supports VM management and resource distribution capabilities, while monitoring support is provided by *Resource Monitor (RM)*.

VtM is mainly in charge of managing the VMs lifecycle and their resources during its execution. It firstly creates a customized VM according to the user requirements, and then it manages the resources in order to ensure the QoS and finally destroys the VM. The first task is the VM creation which requires the following steps: downloading and creating the guest operating system (a Debian Lenny through debootstrap for this prototype), copying extra software needed by the client in an image that will be automatically mounted in the VM, creating home directories and swap space, setting up the whole environment, packing it in an image, and starting the VM. Once the VM has completely started, the guest operating system is booted. After this, the additional software needed by the client needs to be instantiated (if applicable). These phases can be clearly appreciated in the evaluation section.

From this description, one can derive that this process can have two bottlenecks: the network (for downloading the whole guest system) and the disk (for copying applications and creating system images: nearly 1GB of data). The network bottleneck has been solved using a caching system which creates a default image of the guest system with no settings, and copying this image for each new VM. This almost eliminates the downloading time (base system is downloaded once and it is reused for each new VM), but contributes to the disk bottleneck. The disk bottleneck has been solved by adding a second caching system that periodically copies the default image and the images with the most commonly used software to a cache space. Finally, these images have only to be moved (just an i-node change) to the final location when a new VM is created. Using both caching systems, the complete creation of a VM has been reduced from up to 40 seconds to an average time of 7 seconds.

In order to add fault-tolerance to the system, *VtM* periodically is able of making checkpoints of VMs, which may require this mechanism, and re-take their execution if the node they were running fails. The system is capable of differentiating read-only from read-write parts in the VM image thanks to the use of Another Union File System (AUFS) in the VMs. In this way, only checkpoints of modified parts are performed, thus reducing the time needed to make a checkpoint and, as a consequence, the interference on task execution. These checkpoints are compressed (only if this permits saving time) and, in conjunction with the read-only parts are stored in the previously presented checkpoint storage. The checkpointing mechanism has been fully described in [7].

Once the VM is running, *VtM* can submit tasks into the VMs by using the SSH wrapper that is able to check the status of a task and while this execution is being done, *VtM* calculates the amount of resources that a task really requires. This value will be used to decide the resource assignment to this task later on and will be also provided to the *Scheduler* layer, in order to make it aware of the real requirements of each task.

The resource calculation starts from the estimation initially provided by the *Scheduler* and adapts this value depending on the past resource usage of the task. In particular, the algorithm uses the mean of the last 5 values, the mean of the last 60 values, and the total mean. Using these values, the algorithm distinguishes three situations regarding the resource usage of the task: a) it is rapidly increasing, b) it is normally increasing, and c) it is decreasing. In the first case, the algorithm avoids assigning too many resources to a given task in a small period. In the second case, it provides resources to the task immediately. In the last case, it slowly unallocates resources to the task, giving time to confirm the falling trend before subtracting too many resources.

Once the resource requirements estimation has been done, *VtM* decides the amount of resources to be allocated to each VM. In addition, surplus resources of the node can be distributed among the running VMs of a node according to different policies and it recalculates the resource assignment of the VMs every time that either the calculated resource usage or the *Scheduler* requires it. Finally, when the VM has finished its taks, *VtM* destroys the VM and stores the required information in the system storage. Further details of all these mechanisms can be found at [8] and [9].

When dealing with local nodes in the provider, these functionalities are provided by *VtM*. However, when dealing with remote nodes in a federated Cloud provider, these functionalities are provided by *Virtualization Manager External (VtME)*, which allows renting VMs in an external provider such as Amazon EC2 and deploying tasks within them as if done locally. Nevertheless, *VtME* provides less functionality than *VtM*, because the control over external resources is lower than over local ones, which makes capabilities such as efficient migration or checkpointing not feasible. *VtME* also provides facilities to offer unused resources to other providers.

Resource Monitor provides a mechanism for retrieving information of the resources and provides this information to upper layers or external components. This information contains different some dynamic metrics such as the CPU and memory usage or others like the architecture of the VM or its number of CPUs. It gets the information from the different resource managers and provides a Ganglia-like XML file with this information. Furthermore, it has

a cache system in order to avoid possible overstress of the monitoring system.

Nevertheless, *RM* not only provides information about the VMs but it is also in charge of resource discovery and topology maintaining. It periodically checks if new nodes has been added to the system and informs the *Scheduler* layer about this. In addition, it also provides failure management by periodically checking if resources are still available and notifying the upper layer if any of them breaks down.

V. Scheduler

The last EMOTIVE layer is *Scheduler* which is in charge of merging all the nodes and abstracting them as a single big resource where VM are running without taking into account if this VM is running in a given node or it is migrated from one to another. As it has been already presented, the *Scheduler* layer is intended to be a wrapper for different schedulers with different features and capabilities that can take advantage of EMOTIVE common facilities such as migration, resource discovery or checkpointing.

Thanks to the system capabilities, the *Scheduler* takes advantage of the power of bottom layers such as the creation of VMs in an efficient way in any of the node. Another capability is migration; it can decide to migrate a VM from a node to an other in order to re-locate its resources. In addition, if a node that was executing different VMs breaks down, the *Scheduler* can decide to recover its execution.

In terms of topology maintenance, the *Scheduler* can take advantage of the *RM* capabilities and it only needs to implement the VRMMScheduler API which includes the retrieving the nodes in the system, getting the location of a VM or task and finally the adding and removing of nodes. Implementing these methods, the *RM* will be able to notify if a node managed by the scheduler is up or down and allows monitoring different metrics of the nodes.

Different implementations are already developed such as the *Semantically-enhanced Resource Allocator (SERA)* [9] which adds semantics for describing the resources and offers a resource allocation based on these descriptions. EERM [10] is another implementation that provides economic enhanced resource allocation according both to the market requirements and the state of the resources.

Examples of schedulers currently in development are the *Federated Scheduler (FEDS)* which decides the allocation of the services in the nodes guaranteeing to each one enough resources to meet the agreed performance goals and trying to maximize provider's utility and includes allocating additional resources from other Cloud providers. Other plans includes the development of a scheduler that makes use of non-deterministic techniques, such as the Simulated Annealing approach, for making application and VM placement decisions, as well as the use of the MapReduce programming model as a mechanism for distributing and orchestrating the process of distributedly collecting performance monitoring data, optimizing application placement and enforcing application and VM reconfigurations.

TABLE I
Time to create a Virtual Machine using different approaches

Action	No	1 level	2 level
Download base system	88.1	-	-
Create base system image	68.4	-	-
Copy base system image	-	45.2	2.3
Copy software image	13.9	13.9	0.0
Create home & swap	13.7	13.7	0.0
Load image	4.4	4.4	4.4
Total time for running	184.6	77.2	6.7

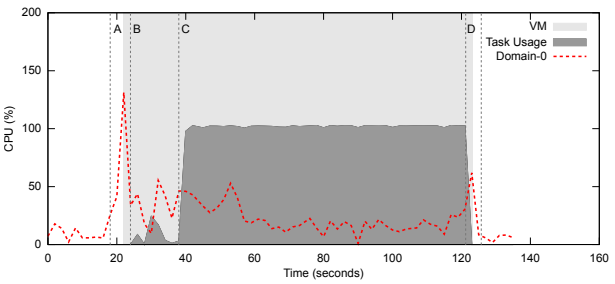

Fig. 2
One task lifecycle

VI. Evaluation

A. VM creation performance

This section provides some indicative measurements about the time needed to create a VM and make it usable for a customer's application, and the benefit of our cache systems for reducing this time compared with the default approach. Table I shows the times required to perform each one of the stages for creating a VM detailed in Section IV. It shows that thanks to the two caching levels the time for creating a new VM can be near to 7 seconds.

Nevertheless, the above VM creation time does not include the time needed by the guest operating system to boot and be available to the user. This time can be appreciated in Figure 2, which shows the CPU usage of a given VM that executes a task during its whole lifetime (from the VM creation to the VM destruction), including also the CPU usage of the Xen Domain-0.

As shown in Figure 2, during phase A, the Xen Domain-0 creates the VM. This spends almost one CPU. During phase B, the guest operating system is booted (first peak in the CPU usage graph). At this point, the customer's task is executed during phase C. Finally, during phase D, the Xen Domain-0 destroys the VM. Notice that the CPU consumption of the Xen Domain-0 is only noticeable during the creation and destruction of the VM. The results in this figure confirm that the creation of a VM takes around 7 seconds and according to this while the full creation of the VM takes around 20 seconds.

B. VM placement

This section demonstrates how resources are dynamically reallocated among applications according to the calculated resource usage of each application. The experiment consists of running a total amount of five tasks with different resource requirements in a provider with two nodes.

Figure 3 shows the CPU allocation and the CPU usage of these tasks in the two hosts of the provider, as well as the CPU usage of *VtM* (which runs in Domain-0). When *Task 1* is submitted to the system with a CPU requirement of 300%, it is scheduled in *Host B* and when *Task 2* is submitted, it cannot run in *Host B* because it needs 150% of CPU and there is only 100% of CPU available in *Host B*; therefore, it is executed in *Host A*. Notice that, since *Task 1* and *Task 2* are the only tasks running in *Host B* and *Host A* respectively, they get allocated all the resources in their host.

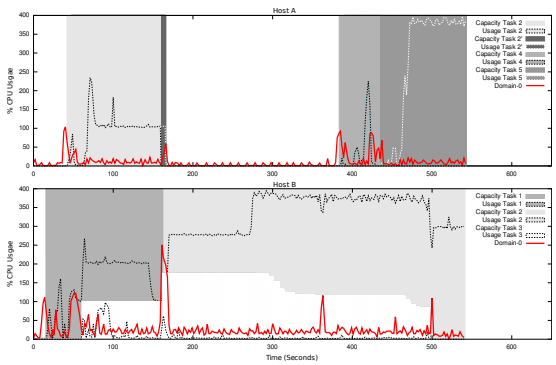

Fig. 3
TASK PLACEMENT AND CPU ALLOCATION/USAGE

When *Task 3* arrives at the system, it can be scheduled in *Host B* and both *Task 1* and *Task 3* share the resources in this host according to their requirements and following the *VtM* resource assignment. Once *Task 1* finishes, 300% of CPU at *Host B* becomes free, and according to its configured policy *Scheduler* migrates *Task 2* to this host. A new domain (*Task 2'* in the figure) contains the task that is being migrated. Notice that our data management implementation using NFS (see Section III) allows migrating VMs very efficiently and makes this always available to the clients in spite of being migrated. Additionally, notice that the CPU consumption of Domain-0 is increased during the migration and this must be considered by the scheduling policies.

Finally, when *Task 4* is submitted, it is scheduled in *Host A*, since there are no enough free resources in *Host B*. Once it executes, it simply returns and the VM is destroyed. At this point, *Task 5* is scheduled in *Host A* and all the CPU in this host is assigned to this task. Later, *Task 3* finishes and *Task 2* and *Task 5* keep running in *Host B* and *Host A* respectively, since these tasks are unfinished.

It demonstrates the capabilities of the EMOTIVE architecture for making scheduler able to relocate VMs easily inside the provider with no extra effort since it relies on the bottom layers.

VII. RELATED WORK

Grid computing as a platform for developing distributed applications has not been so successfull as it was hoped and one of the main reasons is the low level of abstraction this paradigm required. [11] establishes how Clouds can be viewed as a logical and next higher-level abstraction from Grids by providing a higher-level of abstraction. This concept has been widely discussed in recent times, [12] presents some key concepts of this paradigm such as the illusion of infinite computing resources available on demand and the ability to pay for use of computing resources on a short-term basis as needed. This allows companies to have an small set of resources that can be increased according to their needs saving costs.

Virtualization is probably one of the main innovations of Cloud and the one will cause a major transformation of the IT infrastructures in the coming years [13]. Among all the Virtualization technologies those which gives the desirable level of performance for working as a platform for Cloud computing are full-virtualization and paravirtualization. Full-virtualization uses a virtual machine that mediates between guest operating system and the native hardware like VirtualBox [14], and VMWare [15]. Paravirtualization also uses a hypervisor but integrates some virtualization parts into the operating system obtaining a better performance but implying a modification on the guest OS. Some of the most famous examples of paravirtualization are Xen [2] and KVM [16]. Finally, a sign of how important virtualization is becoming is the addition of hardware capabilities that support this [17].

There are different alternatives that provides a Cloud solution relying on VMs and probably the most popular is Amazon EC2 [18] which allows the user having a VM where executing his jobs. Nevertheless, it is a private implementation and it does not allow working with low-level aspects. In order to avoid this problem, different Cloud implementations which implements the EC2 API such as Eucalyptus [19] or Nimbus [20] have appeared. Other open-source alternatives such as OpenNebula [21] also adds outsourcing capabilities by adding external resources. Nevertheless, this typology only works with a given pool of images of VMs with different capabilities that are instantiated when the user wants to use them. Nevertheless, our approach creates a customized image from scratch according with the user requirements in an efficient way.

In addition, these approaches make a static allocation of the resources by assigning a given amount of resources at the beginning of the VM creation. However, we provide a mechanism that re-schedules the resources of each VM according with the SLA agreed with the user. This allows a more efficient use of the resources without any underusage while respecting the terms agreed with the user.

Introducing virtualization for abstracting nodes of a service provider and allocating tasks in a VM in order to consolidate and isolate them inside the same physical machine has been widely investigated during the last years. Some of the proposed solutions are driven by SLAs, as in our case, like Oceano [22] which take into account the customer needs.

Lately, some works have exploited virtualization capabilities for building their solutions. It has been used to facilitate system administration and provide the users with dedicated and customized virtual working environments, making more comfortable their work like in the case of VMShop [23] or other works like Globus Virtual Workspace [24] and SoftUDC [25] which use virtualization features such as pausing and migration.

VIII. Conclusions and future work

In this paper, we have presented a complete stack of software that permits implementing a Cloud solution which simultaneously ensures an efficient usage of the local and the remote resources. We propose an approach that exploits the advantages of virtualization for accomplishing resource distribution as well as application lifecycle management.

EMOTIVE Cloud offers a platform to developers for working with a Cloud computing implementation that takes advantage of the virtualization capabilities and provides a transparent management of the resources and allows optimizing the provider resources. Moreover, it can be used in order to implement a private Cloud provider and extend it taking advantage of all its features. Our evaluation has demonstrated the functionality of the proposal and the efficiency of the system by offering different features such as dynamic VM creation and migration.

Our future work includes an improvement of scheduling policies which will include economic indicators and intensively exploit our architecture capabilities such as migration and checkpointing. Finally, we also plan working on managing mixed sets of heterogeneous workloads, comprising transactional applications, CPU and memory intensive jobs, and data-driven MapReduce tasks.

Acknowledgement

This work is supported by the Ministry of Science and Technology of Spain and the European Union (FEDER funds) under contract TIN2007-60625, by the Generalitat de Catalunya under grant 2009FI B 00249, and the European Commission under FP6 IST contract 034556 (BREIN).

References

[1] "EMOTIVE Cloud," `http://www.emotivecloud.net`.

[2] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield, "Xen and the art of virtualization," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, New York, NY, USA, 2003, pp. 164–177, ACM.

[3] E. Mellor, R. Sharp, and D. Scott, "Xen Management API," Revision 1.0.6. July 2008.

[4] "libvirt," `http://libvirt.org`.

[5] I. Goiri, F. Julia, and J. Guitart, "Enhanced Data Management for Virtualized Service Providers," *Proceedings of the 17th Euromicro Conference on Parallel, Distributed and Network-based Processing*, pp. 409–413, 2009.

[6] D. Borthakur, "The Hadoop Distributed File System: Architecture and Design," *Document on Hadoop Wiki*, 2008.

[7] I. Goiri, J. Julia, J. Guitart, and J. Torres, "Checkpoint-based Fault-tolerant Infrastructure for Virtualized Service Providers," in *Submitted to 22nd ACM Symposium on Operating Systems Principles (SOSP'09), Big Sky, MT, USA, October 11-14*, 2009.

[8] I. Goiri, F. Julia, J. Ejarque, M. De Palol, R. M. Badia, J. Guitart, and J. Torres, "Introducing Virtual Execution Environments for Application Lifecycle Management and SLA-Driven Resource Distribution within Service Providers," *Proceedings of the 8th IEEE International Symposium on Network Computing and Applications*, 2009.

[9] J. Ejarque, M. De Palol, I. Goiri, F. Julia, , R. M. Guitart, J. Badia, and J. Torres, "SLA-Driven Semantically-Enhanced Dynamic Resource Allocator for Virtualized Service Providers," *Proceedings of the 4th IEEE International Conference on eScience (eScience 2008), Indianapolis, Indiana, USA, December 7-12*, pp. 8–15, 2008.

[10] Mario Macías, Omer Rana, Garry Smith, Jordi Guitart, and Jordi Torres, "Maximizing revenue in Grid markets using an Economically Enhanced Resource Manager," *Concurrency and Computation: Practice and Experience*, vol. 9999, no. 9999, pp. n/a, September 2008.

[11] S. Jha, A. Merzky, and G. Fox, "Using clouds to provide grids with higher levels of abstraction and explicit support for usage modes," *Concurrency and Computation: Practice and Experience*, vol. 21, pp. 1087–1108, 2009.

[12] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al., "Above the clouds: A Berkeley view of cloud computing," *University of California, Berkeley, Tech. Rep*, 2009.

[13] M. Rosenblum and T. Garfinkel, "Virtual machine monitors: Current technology and future trends," *Computer*, vol. 38, no. 5, pp. 39–47, 2005.

[14] J. Watson, "VirtualBox: bits and bytes masquerading as machines," *Linux Journal*, vol. 2008, no. 166, 2008.

[15] "VMWare," `http://www.vmware.com`.

[16] "Kernel-based Virtual Machine (KVM)," `www.linux-kvm.org`.

[17] K. Adams and O. Agesen, "A comparison of software and hardware techniques for x86 virtualization," in *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*. ACM New York, NY, USA, 2006, pp. 2–13.

[18] Amazon, "Amazon elastic compute cloud," `http://aws.amazon.com/ec2`.

[19] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-source Cloud-computing System," *Proceedings of Cloud Computing and Its Applications*, October 2008.

[20] "Nimbus Science Cloud," `http://workspace.globus.org/clouds/nimbus.html`.

[21] "Opennebula," `http://www.opennebula.org`.

[22] K. Appleby, 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, vol. 5, pp. 855–868.

[23] Ivan Krsul, Arijit Ganguly, Jian Zhang, José A. B. Fortes, and Renato J. Figueiredo, "Vmplants: Providing and managing virtual machine execution environments for grid computing," *SC Conference*, vol. 0, pp. 7, 2004.

[24] K. Keahey, I. Foster, T. Freeman, and X. Zhang, "Virtual workspaces: Achieving quality of service and quality of life in the grid," *Sci. Program.*, vol. 13, no. 4, pp. 265–275, 2005.

[25] M. Kallahalla, M. Uysal, R. Swaminathan, D.E. Lowell, M. Wray, T. Christian, N. Edwards, C.I. Dalton, and F. Gittler, "SoftUDC: a Software-based Data Center for Utility Computing," *Computer*, vol. 37, no. 11, pp. 38–46, 2004.