Use Cases of Virtualization in the Management of Distributed and Parallel Systems

Stefania Costache, Nikos Parlavantzas, Christine Morin, Samuel Kortas

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Use Cases of Virtualization in the Management of Distributed and Parallel Systems

Stefania Costache — Nikos Parlavantzas — Christine Morin — Samuel Kortas

N° 0399

December 2010

Rapport technique

ISSN 0249-0803 ISRN INRIA/RT--0399--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

Use Cases of Virtualization in the Management of Distributed and Parallel Systems

Stefania Costache* † ‡ , Nikos Parlavantzas*‡ , Christine Morin*‡ , Samuel Kortas†‡

Thème : Systèmes et services distribués
Équipe-Projet MYRIADS

Rapport technique n° 0399 — December 2010 — 33 pages

Abstract: Efficiently sharing resources between multiple applications that run on the same distributed infrastructure is challenging. These applications have different and possible time varying requirements and in the same time, users demand different quality of service guarantees. However, current resource management systems still make it difficult to specify and meet these requirements. Recently, virtualization technologies have been seen as a useful tool for managing distributed infrastructures. This lead us to survey the use cases of virtualization in distributed computing. We start from giving an overview of virtualization techniques together with what advantages and concerns may come from their use. Then we introduce the issues that may come from managing distributed virtualized infrastructure. We present a taxonomy of representative works that used virtualization as a tool for resource management. Finally, we conclude with an analysis of the advancements done in resource management with the use of virtualization.

Key-words: Virtualization, Resource Management, Distributed Systems

This work was done in the context of the CIFRE colaboration EDF-INRIA N° 03322010

* INRIA Rennes-Bretagne Atlantique, Rennes, France
† EDF R&D, Paris, France
‡ Stefania.Costache@inria.fr, Nikos.Parlavantzas@irisa.fr, Christine.Morin@inria.fr, Samuel.Kortas@edf.fr

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Les cas d'utilisation de la virtualisation dans la gestion des systèmes parallèles et distribués

Résumé : Le partage efficace des ressources entre plusieurs applications qui s'exécutent sur la même infrastructure distribuée est un défi. Ces applications ont besoins différentes et dans le même temps, peuvent avoir des exigences différentes, comme la qualité de service. Toutefois, les systèmes actuels de gestion des ressources font qu'il est encore difficile pour les utilisateurs de spécifier leurs exigences et d'obtenir une réponse adéquate. Depuis peu, les technologies de virtualisation sont considérées comme des outils utiles pour la gestion des infrastructures distribuées. Dans ce rapport, nous examinons les cas d'utilisation de la virtualisation dans les systèmes distribués. Nous présentons une étude des techniques de virtualisation et les avantages et inconvénients qui proviennent de leur utilisation. Ensuite, nous décrivons les problèmes qui peuvent provenir de la gestion des infrastructures virtualisées distribués. Nous présentons une taxonomie des oeuvres représentatives qui utilisent la virtualisation comme un outil pour la gestion des ressources. Enfin, nous concluons avec une analyse des progrès effectués dans la gestion des ressources avec l'utilisation de la virtualisation.

Mots-clés : Virtualisation, Gestion de Ressources, Systèmes Distribués

1 Introduction

Computational infrastructures, like clusters, grids or clouds, are used by a variety of users to run applications that require large amounts of resources. These applications can come in different types and with different resource demands. Also, users could request different guarantees for their execution. Some users could require to have their applications executed before a deadline while others just want to have resources for their applications available with reasonable delays. These different scenarios raise the problem of how to efficiently share the same infrastructure between multiple applications to meet all these demands while in the same time maximizing the resource utilization.

Job based management systems, such as SGE [12], Moab [39] or Maui [37], were designed to accomodate the execution of multiple applications on a shared computational infrastructure. These systems provide to users the capability to run their applications on statically allocated resource slots, i.e. nodes, and for a specified amount of time. After the resources are allocated for the job, the job's tasks are statically mapped to the nodes. Thus, the user needs to estimate the amount of resources and the time required by the job to finish its execution. However, using only user estimates is not enough because they can lead to resource underutilization and bad performance of the application. The first case happens when the user allocates too many resources for the application and/or the running time is overestimated. The second case happens when allocated resources are not enough and/or the running time of the job is underestimated. To overcome this rigidity, some batch schedulers added support for advanced customizable policies and priorities for applications. But, these are globally defined and involve a lot of manual tuning for administrators. Another encountered problem is that certain user constraints are difficult to meet in such systems. One common use case for this is the execution of deadline-driven applications. Such applications need to provide results until a specific deadline, so they could be useful. However, considering the resource volatility and the variety of user demands, they cannot benefit from predictable running times. To overcome this issue, advanced batch schedulers have support for reserving resources in advance at specified times. But not only that this involves the user to plan its reservation, it also leads to severe resource underutilization [36]. These problems illustrate the fact that such provisioning mechanisms are not flexible enough to meet the guarantees users want for their applications and to efficiently manage resources.

Virtualization provides the capability to partition physical machines in multiple virtual machines, each of them having its own software stack. Moreover, virtualization technologies allow to suspend, resume and migrate the virtual machine transparently for the application that runs inside it. These abilities made it adopted by multiple groups as a mechanism to address the various issues present in the traditional mechanisms for managing shared infrastructures. In the HPC and grid context, it was seen as a mechanism to allow users to configure their own environments and to improve resource utilization by overcoming the software heterogeneity of different infrastructures and improving traditional resource provisioning mechanisms [14, 11]. In datacenters, virtualization was used to provide infrastructure on demand to users and charge them for its use [2]. It was also adopted to isolate multiple applications, like multi-tier web servers,

and to consolidate them on fewer servers, maximizing the resource utilization and leading to power and energy savings.

In this report we investigate the ways that virtualization was used in distributed and parallel computing, to see how the flexibility of virtualization technologies can be used by resource provisioning mechanisms to meet user constraints and improve resource utilization. Our objective is to give an overview of the recent work that adopted virtualization for managing shared infrastructures, identify the main covered directions and classify these approaches. This report is structured as follows. In Section 2 we give an overview of virtualization technologies and we describe what advantages the adoption of virtualization in distributed computing can bring, together with the concerns that can rise from its use. In Section 3 we introduce the problem of managing virtualized infrastructures and in Section 4 we present a taxonomy of the solutions that use virtualization as a resource provisioning mechanism. Finally, we discuss at what extent current solutions address our problem in Section 5.

2 Virtualization in Distributed Computing

Virtualization started to gather more and more interest in distributed computing. The notion of cloud computing introduced virtualization as a provisioning mechanism and public clouds like Amazon EC2 [2] use this model to provide users with virtual machines on demand and for specified prices. Also, shared testbeds like PlanetLab [4] and Emulab [23] adopted virtualization as a mean to provide better resource control of their infrastructure and isolation between users. In this Section we give an overview of virtualization technologies and what capabilities they offer. Afterwards we present what advantages and concerns the adoption of virtualization in distributed computing brings. Finally we conclude with outlining main directions that can be found in recent research work that uses virtualization in distributed computing.

2.1 Overview of Virtualization Technologies

Virtualization allows multiplexing the physical resources between multiple virtual machines. A virtual machine can emulate a physical host by having its own memory state, disk and device configuration, independent from the host system. Applications run in a virtual machine as if they were running on the native system. A hypervisor manages all the virtual machines from the same host and controls their access to resources. Based on their implementation, virtualization technologies were classified in four different types:

Full virtualization Full virtualization technologies provide complete hardware emulation such that operating systems from virtual machines run unmodified, like on real hardware. As operating systems from the virtual machine need to issue privileged instructions to configure system resources, the hypervisor needs to intercept and translate them. Otherwise they will fail because they will not execute in the right privilege level. These actions bring a big performance penalty on the applications running in the virtual machines. VMWare Server [70] or Virtual Box [66] are examples of full virtualization technologies.

Paravirtualization Paravirtualization technologies are also based on a hypervisor that runs on top of the bare hardware but the hardware is not emulated. In this case, the operating system running in the virtual machine is modified so that its access to resources is done through the hypervisor interface. A disadvantage of paravirtualized systems is that the guest operating system must be modified so that it can interact with the hypervisor. Xen [3] is an example of paravirtualized system.

Native virtualization To make virtualization easier, hardware extensions have been added to recent processors. These extensions allow virtual machines to run their privileged their instructions at the right privilege level, with the hypervisor intercepting only certain instructions. In this way, operating systems can be run without modifications and without having the overhead of full virtualization technologies. KVM [30] and recent versions of Xen is an example of hypervisor that takes advantage of hardware virtualization.

Operating system-level virtualization This technique isolates applications in separate containers that share the same operating system. The hypervisor in this case is the host operating system kernel. This provides much better performance than the previous virtualization technologies, as the access to hardware resources is done by the same host operating system. Moreover, fewer resources are consumed, as there is only one operating system. However, containers do not provide the same isolation guarantees as the other virtualization technologies. As the operating system kernel is shared, it is easier to compromise it and also the other containers. Well known container-based virtualization technologies are OpenVZ [47], LXC [35], or Linux VServers [67].

Virtualization technologies come with a number of attractive features. Hypervisors usually provide control knobs to adjust CPU, memory, disk and network bandwidth for each virtual machine. Other supported capabilities are to suspend and restore the state of a virtual machine and live migration. These are present in most of virtualization technologies, including operating system level virtualization as illustrated by OpenVZ. These features rely on the fact that the state of a virtual machine composed of its memory content, metadata concerning I/O accesses and CPU registers, can be saved on local storage or transferred across the network to a different node. When the hypervisor suspends a virtual machine, it pauses it then it saves its state. At a later point in time, the virtual machine can be resumed on the same or on a different host. Through live migration [8], the state of a virtual machine is transferred to the destination node without stopping the virtual machine on the source node. The transfer continues until only the frequently accessed memory pages remain. Then, the virtual machine is stopped and the remaining state is transferred to the destination.

However, these mechanisms have currently their own limitations. The resource control mechanisms for virtual machines are dependent of the hypervisor implementation as not all virtualization technologies provide the same control knobs. Then, both suspend/restart and live migration do not handle the virtual machine disk state. For suspend/restart, separate mechanisms to store the modifications are needed while for live migration the state is assumed to be stored in a shared storage between the source and destination node. Finally,

container migration is possible only between hosts that have the same physical configuration and the same operating system kernel.

2.2 Motivations and Concerns of Using Virtualization

Due to the features that it provides, virtualization became of interest to use in distributed computing [14, 25, 64]. In this context, users are provided with abstractions like *virtual environments* or *virtual clusters* that represent sets of one or more configured and interconnected virtual machines. Using these abstractions for managing physical infrastructure resources brings several possible advantages and issues. We describe them in this section.

2.2.1 Motivations

There are several advantages brought by virtualization that were explored by previous work. From these, we identify three main directions of use of virtualization: i) customization and isolation of user environments; ii) transparent fault tolerance for applications; iii) flexible resource control.

Customization and isolation One important use case of virtualization is having distributed virtual private environments that can be customized by need. Many application types can be executed on the same physical infrastructure, as each application can have its own software stack, from operating system to libraries, independent of the physical platform they are running on. Also, applications can be executed on sites that don't necessary have the software configuration required by the application. Because virtual environments are isolated one from another, users can be allowed to execute privileged operations without getting special privileges from system administrators and the malicious user behaviour can be restrained.

Transparent fault tolerance An application running in a virtual environment can be transparently migrated to different hosts, when failures of current nodes are imminent. In this case using virtualization overcomes the need of installing operating system kernel modules or linking the application to special libraries.

Flexible resource control Flexible resource management policies can be designed by controlling the resource allocation of the virtual environments in a fine grain manner during the virtual environment's lifetime. Physical nodes can be time-shared between multiple virtual machines by controlling the access to node's resources like CPU, memory and network bandwidth. In the same time, virtual environments can be seamlessly migrated between physical nodes or clusters when the resource allocation does not correspond to the host capacity or due to administrative decisions (e.g. nodes need to be shut down for maintenance). Various energy management policies can be designed by consolidating the virtual environments on as few nodes as possible. By using virtualization, distributed virtual infrastructures, like clusters and grids, spread over multiple physical clusters at different geographical locations, can be provided on-demand. Priority and load balancing policies can be designed for them in a transparent way, by changing their capacity on-the-fly.

2.2.2 Concerns

The main concern related to using virtualization in distributed computing is the impact on the execution of applications running in virtualized environments, especially for scientific applications as they target optimal execution times. From the application's perspective, virtualization brings a runtime overhead, thus increasing the application execution time. From the resource provider's perspective, virtual environment management could bring an overhead on the overall infrastructure throughput and also leads to wasting resources, as times in which resources can be used for computation are lost on managing the virtual environments.

Application runtime overhead One important concern of running applications in virtualized environments is the performance overhead. Performance overhead can come from: i) operating system noise; ii) device virtualization; iii) inference with other co-located virtual machines; iv) resource management policies involving virtual machine migration.

Operating system noise is produced by both the guest operating system and the host operating system in which the hypervisor usually runs [64]. Device virtualization brings a performance overhead as access to devices is done through the host operating system. For example, an overhead of 17% for communication intensive applications is noticed in [25] by running a NAS Parallel Benchmark in a virtual and a native environment. However, the induced overhead can be much smaller when communication constitutes a small part from the application execution [15]. To deal with this problem, advanced techniques can be used to allow virtual machines to directly communicate with devices [25] and dedicated cores can be used by the hypervisor to process operations on behalf of the virtual machines [19]. For example, in [25] the authors show that by using such a technique the performance overhead for an intensive communicating application running with Xen was reduced from 17% to 4%. However, this overhead was measured with regard to the total execution time of the application. In [52] the authors chose a different measure, by analyzing the effects of sharing an Infiniband network connection between multiple virtual machines hosted on a multicore platform. Their results showed that by separating the components of application in different virtual machines and pinning them to different cores the communication overhead can become negligible. However, such techniques have their own complications. For example, they make operations like checkpoint or migration more difficult to implement, as the hypervisor does not have access to the entire virtual machine state anymore.

Performance interference is caused by placing multiple virtual machines on the same physical machine. Because virtual machines share the same resources of the physical machine, running an application in virtual machines is different from running it on a physical machine with the same capacity. For example, even if CPUs are pinned to virtual machines, there is still CPU cache interference that can lead to the performance degradation of the applications [32]. Also, the hypervisor needs to manage the multiple virtual machines and so, it adds a processing overhead.

Resource management policies can involve the migration of virtual environments between different nodes or clusters. Migration not only consumes network bandwidth between the source node and the destination node, but also CPU

and memory. This has an impact on the runtime of the application that runs in the virtual environment, and also on the other applications from the system.

Virtual environment management overhead Managing virtual environments involves several steps that can add an overhead on system's resource utilization. To run applications in virtual environments, virtual machines need to be deployed on the required nodes and started. Also, when the application finishes its execution, the virtual machines need to be stopped. More flexible resource management policies also involve suspending and restarting the virtual machines, and migrating them on different physical nodes. Each one of these steps adds its own overhead, that can be different, depending of the workload type. This was shown in [10]. The authors wanted to improve the resource utilization by running jobs in virtual machines across multiple clusters, when resources from one would not have been sufficient. However, for transactional workloads (jobs with really short run times), the results showed that the management of virtual machines brought a high penalty on the total processing time. This penalty came from the time to instantiate and destroy the virtual machines, that was actually higher than the job runtime. In the case of long-time jobs, improvements were noticed as resources were better utilized. This shows that policies of managing virtual machines should be different for each type of workload, and thus for each application type.

2.3 Summary

Virtualization came as a method to create multiple isolated environments that share the same physical machines. This was complemented further with the capability of suspending and restarting each virtual machine, and thus, the user's entire environment. These features made virtualization to be considered as a tool for managing physical infrastructures. Two different concerns came from this approach, as they were outlined in recent research work: the runtime overhead brought by running applications in virtual machines and the management overhead of virtual machines. However, even if these concerns are not negligible, they do not lead to excluding the use of virtualization in distributed and parallel computing.

Different research groups focused on how to make use of virtualization to improve the management of distributed infrastructures. We can identify different directions in their work. A first direction is the management of virtual machines, to provide virtual clusters on demand to users. This case uncovered several issues, like minimizing the creation and the deployment time of multiple virtual machines or managing virtual machines at runtime across multiple sites. Another direction is to improve the life quality of user's virtual environments with fault tolerance mechanisms. Finally, a different focus is on extending or devising new resource provisioning mechanisms capable of managing the physical infrastructures resources by using virtualization as a tool for resource control. These mechanisms have the purpose to increase resource utilization and reduce energy consumption by controlling the resource allocation per virtual machine.

3 Virtual Infrastructure Management

We present in this section the work that focuses on managing virtual machines in distributed infrastructures. We introduce the different aspects that were considered and how recent work addressed them.

3.1 Toolkits for Virtual Infrastructure Management

As the use of virtualization raises the issue of efficiently managing virtual machines, multiple virtual infrastructure management toolkits were designed. Theese toolkits provide reusable infrastructures for solving the specific aspects of using virtualization. A generic extensible framework was introduced in [38]. Its goal was to provide basic functionality for the virtual machine management while leaving the development of complex resource management policies at the administrator's latitude through a plugin interface. Other solutions were especially proposed in the context of cloud computing, for Infrastructure-as-a-Service providers. Such examples are Eucalyptus [45], OpenNebula [57] and Nimbus [28].

In Eucalyptus [45] the virtual infrastructure management is hierarchical. A central server, called Cloud Controller, ensures the system-wide arbitration of resource allocations. Each managed cluster has a Cluster Controller that monitors and schedules the virtual machines on its physical nodes. Each node is managed by a Node Controler that gets the information about node's resources (cpu, memory) and propagates it to the Cluster Controller. The virtual machine images are uploaded by users and stored in a repository, being transferred on the nodes at instantiation time.

OpenNebula [57] manages a virtual infrastructure based on a centralized model. Virtual machines can be requested on a best effort basis, contextualized and deployed on physical nodes according to simple predefined policies. Local capacity can be suplemented during increased workloads, by interfacing with a public cloud. More advanced resource management policies are supported through a lease management system, Haizea [60]. For more complex operations, integration with external managers is provided.

Nimbus Toolkit extends Globus Toolkit 4 with management support for heterogeneous virtual clusters (i.e. composed from virtual machines with different configurations) [28]. Virtual machines can be requested on-demand or managed by local resource managers [16]. For the last case, resource slots are guaranteed by the existing resource managers. When the slot has been obtained, virtual machines are deployed on the nodes from a central repository and the client is notified that the resource allocation has succeeded.

3.2 Configuring and Deploying Virtual Machines

Several groups focused on providing virtual clusters on demand to users. To achieve this goal, a series of steps must be considered. Such steps consist in ensuring the creation and assignment of virtual machines to physical nodes, deploying of virtual machine images on the nodes and managing of virtual machines during their life-time. Each step has its own associated issues.

As in some cases it is cumbersome for a user to install a virtual cluster from scratch or to create it's own virtual machine images, several solutions propose to

automatize the creation and deployment process. In [65] the authors propose to manage the creation and deployment of a virtual cluster by extending an existing configuration tool for physical clusters, Oscar. This is done by first installing and configuring the head node of the cluster. Then, images for the cluster compute nodes are created on the head node and transferred on the compute nodes. Thus, to install the virtual cluster only the image for the head node is required. In the In-Vigo [1] middleware the virtual machines are customized according to a given user specification based on a tool called VMPlant [33]. The VMPlant tool allows a client to specify its configuration actions and the order between them through a directed acyclic graph representation. At instantiation time, these actions are applied to the virtual machines in the specified order and only afterwards the virtual machines are deployed on the physical nodes. In [24] the customization of virtual machines is done through a cluster configuration tool and a package caching service, thus avoiding to use user-provided customized images.

Contextualization is another issue that is considered when managing virtual environments. Contextualization is defined as modifying the configuration of a virtual appliance, i.e. the combination between an application and the virtual environment it is running in, at deployment time such that it fits its deployment context. This is required when different configuration parameters are dependent of the site on which the virtual appliance runs and they cannot be known before the virtual machine deployment. For example, such parameters can be IP addresses of the virtual machines or of an existing service provided by the site. The contextualization of a virtual appliance is done by patching the associated VM disk images with a configuration file that specifies a set of contextualization actions. Then, these actions are applied to every virtual machine after its deployment [6].

Minimizing the deployment and instantiation time of virtual machines is a separate issue that needs to be considered. Creating a virtual machine from scratch and booting it requires a significant amount of time. Also, deploying a virtual environment on its allocated nodes is costly as it takes an amount of time proportional with the virtual machine image. The deployment time can be minimized through caching techniques. Such techniques can transfer the virtual machine image on the nodes before the time that the user requested its resources, in case a queue based resource management system is used [58]. Other methods propose to store virtual machine images in a cache directory on the nodes and re-use them for other user applications [10].

To minimize the instantiation time of a virtual cluster, in [34] the authors propose a model to create virtual machines on-the fly by cloning them from a master virtual machine. This avoids the issue of having to create virtual machines from scratch and deploy them on nodes by starting from the assumption that usually virtual machines from the same virtual environment share the same initial application state. Thus, there is no need to copy the entire virtual machine at instantiation time. The cloning of a virtual machine can be done by transferring only a minimal initial state from the parent virtual machine. The rest of the state can be transferred on-demand, by multicasting it to all virtual machine clones. However, this technique required to modify applications running in virtual machines to use a special API for virtual machine instantiation.

3.3 Managing Virtual Machines across Multiple Sites

Managing virtual machines at different sites can be difficult. Besides the previously mentioned issues, other concerns come from migrating the virtual machines between separate sites and ensuring the network communication between them. Transferring the virtual machine state between different sites brings higher overheads, due to the limited network bandwidth that is available. Moreover, as no shared storage is assumed, the disk state of the virtual machine must be managed also. Ensuring communication between virtual machines from different sites is important, as each site can apply it's own administrative policy that can change the network settings of the virtual machines (e.g. IP address assignment) and block the communication(e.g. firewall rules). The network settings of virtual machines must be kept even when changing its physical location. Several solutions proposed to use virtual networks between virtual machines to keep the communication between them.

In [10] and [18] virtualization is used to manage jobs in multi-cluster or grid environments. In [10] jobs are managed through the Moab scheduler. The system allows load balancing between different clusters by forwarding jobs from the loaded cluster queue to the other cluster queues. Jobs that require resources above the capacity of one cluster can span over multiple clusters. This is achieved by the cluster scheduler to which the job was submitted by taking nodes from the control of the other cluster schedulers for the job's lifetime, and returning them when the job finishes. As the authors start from the assumption that all cluster nodes are visible to each other, the network management for virtual machines is done by each scheduler, by keeping a predefined IP addresses configuration and using it when instantiating virtual machines. In [18] the authors propose a system for managing virtual machines at grid level, initially designed for improving the life-time of best-effort jobs in a grid. A manager instance on each site handles the virtual machine instantiation and provides an interface for submitting best-effort jobs in virtual environments through local resource managers. Fault tolerance is provided by periodically snapshoting the virtual environments and storing the snapshots on the node hosting the manager instance. This allows the re-deployment of the virtual environment in case of failures or when resources on which a best-effort job is running need to be used by jobs with a higher priority. Network management is handled by creating separate sub-networks for virtual machines belonging to the same job.

Other work focus on providing clusters on demand that can spread on multiple sites. In [56] the authors provide a virtual marketplace to users. Virtual machines are connected to the client's local network through a virtual network that optimizes the communication routes between them after inferring the communication traffic [62]. In [24] the authors introduce a management system for virtual clusters that is using VPN-based network connections between the virtual machines. To deploy virtual clusters, any site can receive a reservation from the user. Then, the respective site performs the deployment of the virtual cluster on its resources and it extends the virtual cluster with resources from other sites. No resource management mechanisms are presented.

3.4 Fault Tolerant Virtual Environments

Several solutions were proposed for providing fault tolerant virtual environments. The main advantage that comes from this is that applications don't need to be linked with special system libraries, making it a suitable mechanism for executing legacy applications. There are two different mechanisms for providing dependable virtual environments: proactive and reactive.

Proactive fault tolerant mechanisms consist in migrating application processes to different nodes whenever current nodes are suspected to be unhealthy. Unhealthy nodes can be detected by using information from hardware sensors, like an increase in temperature. In this case, the live migration feature of virtualization can be used to implement proactive fault tolerance for virtual environments. Using virtualization for proactive fault tolerance was proposed in [42]. Each node hosted a virtual machine that was used as an environment for running application processes. The migration of the virtual machine was triggered by a health monitoring system that received information from the hardware sensors of the node and uses this information to predict if the node fails due to its deteriorating health. checkpointing mechanisms.

Reactive fault tolerant mechanisms are based on checkpointing applications and restarting them after failures. Classic checkpoint mechanisms are classified in: i) application level; ii) user level; iii) kernel level. Mechanisms at application level require the application modification and explicit implementation of checkpointing by users. Mechanisms at user level require linking with checkpointing libraries and raises issues related to open files or sockets that the application may have at checkpoint time. Finally, mechanisms at kernel level require the installation of kernel modules and are dependent on the operating system. To solve these transparency issues, checkpoint of virtual environments was proposed in recent research work.

Parallel checkpoint of virtual machines comes with two problems: i) keeping the network state consistent; ii) minimizing the amount of saved state and the time required to save it. Keeping the network state consistent can be done starting from the assumption that, as parallel applications use a reliable communication procotol, i.e. TCP, it is enough to save the virtual machine state before a TCP connection timeout occurs. The time required to save the virtual machine state is also important, as during this time all the computation is frozen so that the virtual machine image is saved to disk. The virtual machine state is much bigger than the application state, because it contains the memory state and the disk state. The latter one needs to be saved during an extra step, as current virtualization technologies don't save it during the snapshot.

Several mechanisms were proposed to checkpoint distributed applications that run in virtual machines. Emeneker et al. [9] syncronized all nodes involved in the checkpoint with a NTP server, to save the virtual machine states before any the network connection timeout would happen. Their results show that this mechanism could work for small number of virtual machines, but some issues still need to be addressed. First, their solution is not complete as NTP does not ensure a complete synchronization. Secondly, if servers become overloaded they could answer slower to requests, thus leading to timeouts. Lastly, the disk state of the virtual machine is not considered for checkpoint, although applications could have written data. As a different approach, Hong et al. [46] and Kangarlou et al. [27] show how a hypervisor-based coordinated checkpoint

mechanism can keep the network state consistent by intercepting all Ethernet frames sent during the checkpoint. In both cases a central coordinator is used to initiate the checkpoint/restore through a two-phase commit protocol. In [27] the coordinator also ensures the flushing of in-transit frames between all the virtual machines. Because the frames are processed at restart, the authors are using the assumption that frame transmision is FIFO and reliable. In [27] frame filtering is done during the snapshot to ensure that virtual machines do not receive duplicate or orphan frames. The authors also show that the downtime that the application running in the virtual environment experiences during a snapshot can be reduced through a snapshot method similar to transfering the virtual machine state during live migration. However, even in these two solutions, clock skews can still appear between virtual machines, leading to communication timeouts.

In [68] a different method for coordinated checkpointing of virtual clusters is presented. This time, operating system-level virtualization is used as checkpointer and is integrated with LAM/MPI checkpointing capabilities. In this case, the network state is kept consistent by the MPI library by having application processes waiting for any MPI messages before triggering the checkpoint. Afterwards, when the checkpoint is triggered, the container state is saved together with the filesystem image on the local node, and then replicated on peer nodes.

3.5 Summary

In this section we presented several aspects of managing virtual machines in order to provide users with dependable customized environments, independent of their physical location. Through the development of virtual machine management toolkits, more complex mechanisms can be developed with no need to be concerned about the common functionality (instantiate the virtual machines, monitor them during their runtime and cleaning the state after the virtual machine is stopped). Such mechanisms can be focused on managing a large number of virtual machines and the physical infrastructure on which they are deployed. Several research groups addressed the concerns related to virtual machine configuration, deployment or fast instantiantion. Others investigated the advantages of providing fault tolerant virtual environments to users. However, one problem that remained was the resource provisioning, as all these works either use a simple model, provisioning virtual machines immediately or on a best effort basis, either use existing resource managers. We introduce this problem in the next Section and discuss the related work.

4 Virtualization as a Mechanism for Resource Provisioning

Several research groups focused on using virtualization mechanisms for better resource provisioning. A first use case is to provide virtual infrastructure on-demand. This was adopted in cloud computing to provide users with resources, give them full control over them and charge their resource usage. Resource provisioning is based on an immediate or a best-effort model: the user receives its virtual machines either at the moment of the request, or its request is queued

and fulfilled when resources become available [45, 57, 28]. A second use case is to integrate virtualization with batch systems by extending local resource managers to run user jobs in virtual machines instead of physical resources [10]. However, both models focus more on the capability of maximizing resource utilization by providing customized environments to users, rather than on addressing user guarantees.

Different research works focus on developing mechanisms for resource provisioning using leasing models. Leasing was introduced as an approach for providing a generic provisioning abstraction capable of accomodating all different user demands. A *lease* is defined as a set of virtual machines made available to an user based on a contract that specifies the provisioning terms, like virtual machine resource requirements and the lease duration. Other groups focus on shared virtual infrastructures with dynamic capacity, mapped on the physical ones. In this case, the resource provisioning model is based on policies and negociation protocols to exchange virtual machines between the virtual clusters according to user workload from within each of them. Virtualization is also used in data centers for minimizing the number of used physical servers by consolidating virtual machines on them. These methods consider server workloads characteristics to predict future resource demand and use it for minimizing the probability of breaching existing SLAs (Service-level agreements) when placing multiple virtual machines on a single machine.

We classify these solutions based on two criteria: the degree of isolation offered to the user and the type of resource allocation. Based on the degree of isolation, a virtual environment can be *dedicated* for the user that requested it, or can be *shared* between other users, based on the application type or user's group or organization. Based on the type of resource allocation, a virtual environment can be static, if its resource allocation is not modified during its life time, or dynamic if the allocation changes, due to global resource management policies. Considering these characteristics, existing resource management solutions can provide one of the following: i) static dedicated virtual environments; ii) static shared virtual environments; iii) dynamic shared virtual environments; iv) dynamic dedicated virtual environments. As static shared virtual environments don't bring any important benefit with their use, no efforts have been made in providing them. We give next an overview of the efforts that have been made in each of the remaining directions.

4.1 Static Resource Provisioning for Dedicated Virtual Environments

Static dedicated virtual environments can provide several advantages. Such advantages could be: i) resource allocation flexibility for the users; ii) improved QoS for best-effort jobs; iii) stronger level of autonomy for provider sites. The resource allocation flexibility came from allowing the users to define software environments for their applications that are different than the platform they are running on. A better quality of service was offered to low-priority reservations and jobs by using the virtualization capabilities of suspend/restart to preempt them and save the current computation [60]. Finally, as the virtual clusters provide isolated environments, local resources can be borrowed to external users or organizations in a controlled fashion, allowing provider sites to apply different allocation policies for their physical resources.

To provide dedicated virtual environments, a resource provisioning mechanism like leasing can be used. Different types of leases were defined to accommodate the various user demands. The most encountered ones are best-effort and advance reservation. Best effort leases are scheduled when enough resources are available in the system. Advance reservation leases are scheduled based on their provided starting and ending times, ensuring that there are enough resources available when an application needs them. Advance reservation leases can be used to provide certain guarantees for applications with strict requirements, like having results before a deadline.

Resource leasing models were proposed both at a local level, within a single administrative domain, and for federated infrastructures. A lease-based approach for cluster resource management was proposed in [60, 59] with a focus on supporting different types of leases and handling the overheads that come from virtual machine management [58, 61]. The authors ensured that the time required for virtual machine management is not charged from the lease period, and thus delaying the execution of jobs for which resources were reserved in advance by scheduling the virtual machine deployment before the start time of the lease [58]. Then, in [61] the time required for the suspension or resuming of the lease was also considered in a similar way. Their solution was limited to homogeneous clusters and the suspension of virtual machines belonging to the same lease was not synchronized, thus leading to communication timeouts between the components of parallel applications. Moreover, the estimation of suspend/restart times was done manually and then the determined values were assigned to the scheduler at runtime. As the estimated value was fixed, the variable bandwidth between the cluster nodes was not considered, sometimes leading to wrong estimated times.

Resource leasing across multiple administrative domains was investigated in [26]. The authors propose a solution for resource control called ORCA and a lease management system, Shirako. The main focus of this work was to allow provider sites to keep the local control of their provided resources and to implement different allocation policies in a transparent way for the clients. To achieve this, the resource allocation was based by a two-step lease generation model. The ORCA architecture is based on three types of entities: brokers, site authorities and service managers. The brokers aggregate resources from multiple sites and offer resource leases according to clients requests. The site authorities are in charge of mapping the leases to the physical nodes from each site. Each site authority can apply its own policies, e.g to maximize resource usage by consolidating virtual resources or to avoid faulty nodes. The service managers are in charge of controlling the user's applications and making resource demands on behalf of them. The lease generation was done in two steps [17]. The first step was accomplished by the broker, through assigning tickets to service managers representing the logical mapping of the request to resources from the broker's inventory. In the second step, the service manager presents the ticket to the site authority, which can reject or approve the request. If the request is approved, the site authority selects the physical resources which will be assigned to the lease, instantiates the virtual resources on them and issues the lease.

Lease-based provisioning provides only a generic mechanism to allocate resources to users, but without caring about user's application requirements. In a context in which application specific goals, like optimal execution time, through-

put, response time, need to be accomplished, the burden of managing resources still falls on users. The lease abstraction is not enough to express these goals and it needs to be extended with additional mechanisms. An initial step in this direction was done by Grit et al. by developing a job execution manager that complements the lease-based resource management systems by estimating and requesting an optimal amount of resources for the jobs that it manages [20]. This was done by using an active learning system to predict performance models for an application. The set of training data needed by the learning system was obtained from running the application in different virtual environments with different configurations. However, this method addressed only static resource allocation, making it suitable only for jobs with known resource demands.

4.2 Dynamic Resource Provisioning for Shared Virtual Environments

Several projects have focused on using virtualization as a mean to balance the workload between multiple clusters, borrow resources from external parties in case of high demands or to apply separate policies for different types of workloads. The common goal followed by these projects was to provide users with shared infrastructures with a dynamic capacity dependent of their workload.

We call these infrastructures virtual clusters. Each virtual cluster can support different software configurations and policies as needed by applications and users. A virtual cluster is shared by multiple users, and its resources are generally managed by a batch scheduler, e.g. PBS or Sun Grid Engine. Users submit jobs to the scheduler and don't need to know that they are running on a virtualized infrastructure. Its computing capacity varies with the workload, i.e. the number of jobs in the batch scheduler's queue. The most interesting aspect introduced by this work is the separation of resource management on two levels of control: local and global. At a local level, virtual cluster resources were managed by a batch scheduler installed in the virtual cluster. At a global level, resources were assigned to virtual clusters according to their demand by using some predefined global policies. This separation can ensure more flexibility in resource management, as each local scheduler can follow its own policies and its own goals while the global scheduler ensures a fair share of infrastructure's resources.

Maestro-VC [31] introduced this separation in its architecture, to create on-demand virtual clusters. In case the amount of allocated resources needed to be changed, the local scheduler negotiated a new resource allocation with the global scheduler. If nodes needed to be reallocated to different virtual clusters, the global scheduler notified the local scheduler, such that actions like checkpoint or resize of the virtual cluster could be taken locally.

Different policies or resource management can be found in current state of art, regarding this approach. In [13], the authors proposed to divide a physical cluster in two virtual clusters based on the types of submitted jobs, i.e. serial and parallel. Each physical node can belong only to one virtual cluster at a time. The resource provisioning was based on assigning to each cluster a quota. In case the cluster's quota is exceeded, idle resources can be transferred from the other cluster, by suspending the virtual machines assigned to one virtual cluster and resuming the ones assigned to the other. As no mechanism of checkpointing the parallel jobs is present, the transfer of resources from the serial to parallel

cluster is done by suspending serial job executions for the whole period of a parallel job run.

More general approaches were investigated in [7], [54], [51] and [40]. In [7] the authors proposed to divide a physical cluster in isolated partitions, called virtual clusters, to allow sites to keep full control of their resources while being able to handle requests from different groups of users. Resources were assigned to each virtual cluster by a global cluster manager. The global manager controls the virtual cluster's capacity based on its priority and current workload. To avoid shrinking too much the virtual clusters when one of them has a high demand of resources, predefined minimum sizes were assigned to each of them.

In VioCluster [54] dynamic virtual domains were mapped on physical clusters, with one virtual machine per node. In case of a high workload, a virtual domain could increase its capacity by asking to borrow resources from other known virtual domains. The resource trading was based on a contract between the virtual clusters, defined at lending time. The contract specified the borrowing conditions and then resources are returned based on them.

In [51] the authors showed how different grids can be hosted by the same collection of sites, by isolating them in virtual clusters. Each virtual cluster is associated with a coordinating manager that is in charge of requesting resources when the capacity of the virtual cluster needs to be increased. The management of resources was done by a site broker that gives resource leases to the coordinating managers. The performance of the solution was highly dependent on the lease period: short leases were beneficial for dynamic workloads due a quick resizing of the system while long leases were beneficial for more steady workloads.

In [40] the authors proposed to creation of a virtual cluster per virtual organization, called a Virtual Organization Cluster (VOC). The created VOCs were resized according to the number of jobs submitted by users belonging to the associated virtual organization [41]. The maximum size of a VOC was defined by site administrators, that also specified the site policies to balance the resource usage between the different virtual organizations. Jobs were submitted to the different VOCs through the Condor scheduler. The virtual cluster size was dependent on the number of job submissions, leading to a complete destruction of the cluster in case of no demand. However, this strategy leaded to an increased overhead on the overall job execution time when the demand fluctuates, as the clusters got often recreated.

4.3 Dynamic Resource Provisioning for Dedicated Virtual Environments

Several groups focused on developing methods to maximize the resource utilization by containing applications in virtual environments with time-varying resource allocation assignments. This is mostly the case of datacenters that host multi-tier web applications. As applications do not fully utilize the resources of a physical node all the time, resource utilization can be maximized by placing multiple virtual machines on the same physical machine and ensuring that the resource demand of each virtual machine is satisfied. We describe the work that addressed this problem in this section. We separate the functions of dynamic resource management in three abstract layers and we give an overview

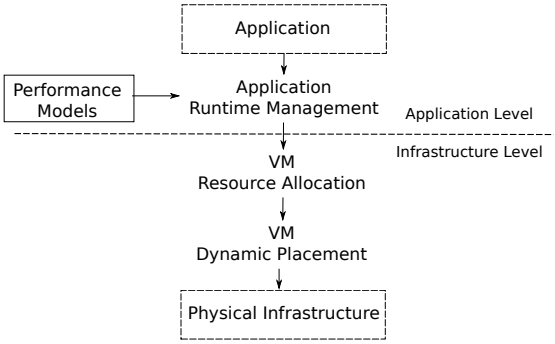

Figure 1: Layers involved in dynamic resource management.

of each of them. Afterwards we describe the main methods to dynamically allocate resources to applications and review some design considerations involved in using them.

4.3.1 The Functions of Dynamic Resource Provisioning

We believe that the resource management process can be decomposed in three layers. As shown in Figure 1, they are: i) application runtime management; ii) resource allocation; iii) resource placement. Application runtime management is a function specific to the application itself and is responsible for translating application requirements to resource demands. The other two layers are dependent on the infrastructure and they work together to meet the provider's objectives. We describe their roles next.

Application Runtime Management The runtime management layer is involved in monitoring the resource utilization of virtual machines and monitoring the application's performance and estimating its resource demands. Monitoring the resource utilization translates to measuring the amount of resources (how much CPU, memory, network bandwidth) that the virtual machine in which the application is running uses. These methods can be classified in black-box and gray-box methods. Black-box methods are used for inferring the resource usage of the application from outside the virtual machine in which is running, while gray-box methods are based on accessing application logs and/or information specific to the operating system running in the virtual machine. Wood et. al. illustrated in [71] the two types of methods to determine the resource usage for a web server application and use this information to avoid overloading servers in datacenters. They concluded that using gray-box methods is better than using the black-box methods in the resource allocation process. However, black-box methods are more desirable for generic resource management platforms, specially when dealing with legacy applications.

Resource demand estimation is usually based on building a performance model of the application. This performance model gives the relation between the amount of allocated resources (e.g. in general CPU) and the application performance metric (e.g. response or execution time). Performance models can be either generated at runtime, or generated offline, by experimentation and

then associated with the application at runtime. For multi-tier web-applications the performance model translates the resource allocation to the response time of received requests. For batch applications the performance model translates the resource allocation amount to the progress the application would make in its computation. Finding a good performance model is difficult, and is also specific to each application. Also, in the case of offline models, the application needs to be run several times on the infrastructure before submission, requiring user involvement.

Virtual Machine Resource Allocation The resource allocation layer is responsible for assigning amounts of resources to the virtual machines in which the application is running. The allocated resource amount can be different from the estimated application demands, as multiple applications compete for the same resources and a fair share between them must be ensured.

Virtual Machine Placement The resource placement layer maps the virtual machines to the physical nodes. In [21] the authors introduce the problem of mapping virtual resources on the physical ones by decoupling the provisioning of virtual machines from their placement on the physical machines. The placement of the virtual machines can change in time, due to various resource management policies. Such policies can target server consolidation or proactive fault tolerance.

Virtual machine consolidation is done usually in datacenters, to minimize the number of used servers [22]. This means to map multiple virtual machines on the same physical node such that their resource demands (amount of CPU, memory, etc.) are satisfied. However, the applications that run in virtual machines have time-varying resource demands. So, the resources allocated to a virtual machine at one time could become insufficient at a different time. Then, the virtual machine would need to be migrated on a different host that can satisfy its resource demand. In case of proactive fault tolerance, the virtual machines are migrated whenever nodes are expected to fail soon. Finally, site administrative policies could impose to migrate virtual machines on different sets of nodes, for example, when maintenance would be needed. In this context, one goal that would be desirable to be achieved is to minimize the overall number of virtual machine migrations between two reconfigurations of the system. This is important as virtual machine migration consumes network bandwidth and CPU at both source and destination nodes and the performance of the application running in virtual machines is degraded during migration. For example, parallel applications are sensitive to variations in network and CPU availability so it would not be desirable to migrate their components too often. As this problem is often associated with consolidation techniques, several other approaches to solve it can be found in this direction [71, 22].

4.3.2 Dynamic Resource Provisioning Methods

Different methods were developed to address the dynamic resource management problem. We identified four types of methods used in literature: i) heuristics; ii) learning algorithms; iii) control theory; iv) utility functions. Table 1 gives a brief overview of them. We summarize the representative related work and how it used these methods in Table 2.

Resource allocation methods	**Advantages**	**Disadvantages**
Heuristics	No need of knowledge about the application	Lack of flexibility; Cannot accommodate all application specific demands
Learning algorithms	No need of previous knowledge about the system or the application	Long training times; Poor scalability
Control theory	Guarantees stability and adapts to changes	Requires precise performance models of the system
Utility functions	Scalability; Capability of expressing how users value their applications	Requires specifying the functions

Table 1: Main state-of-art methods for dynamic resource allocation.

Heuristic algorithms Heuristic algorithms are used to allocate resources to virtual machines based on their utilization level. No knowledge about the application is used but only the information about resource utilization of the virtual machine. To coordinate resource allocation between different virtual machines, priority-based policies are applied.

[74] describes a solution to decentralized fair allocation of resources between multiple virtual machines that compete for a limited resource capacity. Each virtual machine controls its own resource demand and adapts its resource usage by following a rule similar to the TCP congestion protocol. The demand of resources is linearly increased when free resource is available but exponentially decreased in case the resource become overloaded. The results obtained from evaluating the heuristic for web servers sharing the same physical machine show that the performance degradation is much graceful than in the case when no control is applied. However, even if this is advantageous for achieving moderate performance goals while ensuring a fair share of resources, it is not focused on optimizing the performance of applications running inside them.

To provide certain (and different) levels of performance to users, a different resource arbitration scheme was used in [29]. The authors introduced a resource management system that allocates resources to virtual machines according to a set of defined service classes. The service classes describe the expectations users can have from a specified level of service. Each virtual machine has a class that describes the maximum allocation and priority, associated with it. Resources are allocated to virtual machines by a local node controller, based on their class of service. Each virtual machine is started with a maximum allocation according to its defined class, until enough information about its resource usage is gathered. Afterwards, the allocation is decreased linearly but a small buffer, called wiggle room, is kept to account for sudden resource demand. When the amount of a resource is increased, the algorithm also increases the allocation of the other resources from the system with a small amount. When increasing the resource allocation, admission control is applied to limit the increase at the available host

capacity or the virtual machine maximum defined allocation. When not enough resources are available, the virtual machine is suspended. A drawback of this method is that the authors limit the resource usage to the virtual machine class. However, if more resources are available, better classes of service can be offered to users. Moreover, the resource management is limited at only the node level.

Multi-domain resource allocation based on heuristics is presented in [55]. A global entity, called adaptation manager, keeps a view of all the resources from the system and takes decisions about reallocating resources to different virtual machines. Statistics about resource usage are collected on every node by a monitoring daemon, and sent to the adaptation manager in order to adjust the resource usage for each virtual environment and to eventually relocate it on a different physical machine. The amount of allocated resource is adapted incrementally, starting from a predefined range. If the resource usage of a virtual machine is above or below that defined range, the resource allocation is increased. When the host on which the virtual machine resides does not have enough capacity, another host with lower utilization is searched. If no such host is found in the current domain, then the entire virtual environment is migrated to a different domain capable to host it. These two last resource management policies were capable of allocating resources only based on monitoring resource usage outside the virtual machine and with no feedback from the application. Thus, in cases when different higher level requirements are imposed by the users, they could prove too rigid.

Learning-based algorithms Some methods of resource provisioning use machine learning algorithms to estimate the resource demand of an application. These algorithms do not require knowledge in advance about the system, but instead the estimates are obtained through a training phase. They are based on the assumption that there are cases in which the behaviour of the system can be too complex to use analytical models that capture the relationship between multiple inputs and its output. In these cases the relationship between the resource demand and the level of performance of the application is learned at runtime by using resource monitoring information and feedback from the application.

In [53] the authors use a reinforcement learning algorithm to optimize the global CPU allocation for multiple virtual machines residing on the same node. However, their method could be applied only at small scale as reinforcement learning requires long training periods and has scalability issues for large scale problems. In [72] the authors proposed to use fuzzy-logic to learn the relationship between resource usage and application workload and to predict future resource demands. A fuzzy-logic controller transformed the information about past workload and resource usage in a set of IF-THEN rules that are stored in a database and updated whenever new usage patterns appear. Based on this set of rules the controller was able to output the resource demand according to incoming workload. These demands were then sent to a global datacenter controler that allocates resources to optimize the datacenter's profit. However this method had a slow adaptation to sudden variation in workload. A different fuzzy-logic based method is presented in [44], for enforcing application deadline guarantees on top of an IaaS infrastructure. This guarantee is achieved by dynamically provisioning virtual machines from different resource classes (similar as in public clouds) at application runtime. Each application has an associated

agent that monitors its performance and requests more virtual machines or different configurations for existing virtual machines if the current allocation is not enough for meeting a specified deadline. The agent also minimizes the cost of running the job on the cloud infrastructure by relinquishing the surplus of resources. To find out how many resources are required for the job to finish its execution before a deadline, the job is run multiple times with different resource allocations. In this case, the resource allocation and virtual machine placement is done by the IaaS provider, and thus, the used interface only offers limited flexibility.

Control-theory methods Control theory has been recently used in datacenters to control the resource allocation of virtualized environments so that specified SLA for applications is met and the resource usage of the datacenter is optimized [75]. The control of resources is done through a feedback or feedforward loop. In a feedback loop, a controller decides a control input (e.g. amount of CPU cycles) for a supervised system (e.g. a virtual machine) based on the difference between a reference input (e.g. reference application performance metric) and the measured output of the system (e.g. real application performance metric). In the feed-forward loop, the controller uses only the reference input, thus being able to take faster decisions. However, in this last case, the efficiency of the controler depends on having an accurate model of the system and it does not consider any external disturbances that can influence the system behaviour.

Feedback controllers have known a wide adoption as a method for regulating resources for multi-tier web applications in datacenters. In [48] the authors use a feedback integral controller to regulate the CPU allocation for multi-tier web servers running in virtual machines. In the HPC context, control theory was used in [49, 50] as a method to time-share the resources between priority applications that need to complete their execution until a specified deadline and best-effort applications. The method is using an integral feedback controller to allocate as much resource as needed to priority applications such that their progress is sustained and their execution is finished at the specified deadline and not sooner. This method was designed specifically for the case of deadline-driven applications and considered only single-task jobs.

For complex systems, using such controllers is not sufficient as multiple inputs need to be modeled and multiple resources with dependency relations need to be controlled. For such cases, multiple input-multiple output controllers can be developed. However, as relations between multiple inputs and multiple outputs of a system are hard to find, designing them in such cases is difficult. Lastly, controllers need a performance model of an application defined as a linear input-output and they are also application-specific.

Utility-based methods Utility functions represent an attractive way to provide optimal resource allocation for applications. An utility function maps all possible states of the system to scalar values. Associated with applications, utility functions can express the satisfaction that is gained from achieving a certain performance metric (e.g. response time, request throughput). Utility functions can be either provided with the applications, defined by system administrators or users, either learned at runtime. By having each application specify its utility

function, allocating resources to applications becomes an optimization problem defined as finding the resource allocation that maximizes the global utility of the system given by the sum of utilities of all applications under given resource constraints [63]. This is an NP-hard problem and a variety of optimization algorithms from literature can be used to solve it.

As an example, in [5] the authors use an analytical performance model to express the response time of a web server as a function of the number of allocated physical servers and then use this relation to derive the application's utility. To compute the optimal resource configuration they use a beam-search algorithm. To find better resources they use a constraint programming solver to find the resources configuration that maximizes an objective function defined as the difference between application's utility and the cost of redeploying it. In [43] the authors define the optimal global utility of the system as a weighted sum of utilities from all applications and use constraint programming to find the optimal assignment of virtual machines to applications. However, neither of these methods are suitable for large-scale datacenters as having a central algorithm that computes the resource allocation for all the applications running in the datacenter does not scale with a large number of applications and physical resources. In [73] the authors use the utility functions to express the benefit of deploying an application on a set of resources from PlanetLab. Each application has an associated agent that finds a resource allocation that maximizes its objective function defined as the difference between the utility of the application's deployment and the cost of migrating to it. In this case, there is no coordination between the applications, as each of them tries to maximize its benefit independent of the others.

4.3.3 Dynamic Resource Management Design Considerations

We presented in the previous subsections the functions of the dynamic resource management process and how they can be mapped on different layers. We also introduced different approaches to the problem of allocating resources to virtual environments. However, few of them make a clear separation of the resource management functions, and become specific for the problems that the authors try to solve. Most of them follow a two-layer architecture for resource management. Each application has an application agent that handles the runtime management. Basically, the application agent keeps all the application information and makes resource demands based on it. A global arbiter receives all these resource demands and computes a resource allocation for the entire data-center that optimizes its goal. This separation is useful as it encapsulates application specific information in the application agent and simplifies the resource arbiter design. However, having one central entity that manages the resource allocation on behalf of all application types is neither scalable for large scale systems, nor fault tolerant. So the management of resources needs to be distributed between multiple entities that cooperate with each other. From this perspective other solutions propose multi-level and decentralized approaches.

In the multi-level design a global controller has some information about the system and manages the requests at system level. It can take decisions to optimize the global system configuration and can apply system-wide policies. However, a part of management can be done by local controllers, per sets of nodes or/and per node. A node controller decides how much of its resources to

allocate to each virtual machine that is hosted by the physical node. The nodes can then be grouped and managed by an upper-level controller that takes the decisions to provision virtual machines to applications and to distribute these virtual machines between nodes. Such model was applied in [76] by introducing an architecture based on three types of controllers that work at different time scales. In this model, the node controller focuses only on satisfying the performance objectives of the application components that are hosted by it. Multiple nodes are then grouped in a "pod", managed by a controller that adjusts the placement of the virtual machines by following specific placement policies (e.g. for consolidation). Finally, a global controller estimates future resource demands for all applications running in the datacenter and takes load balancing decisions to assign nodes to "pods" and to place applications.

In a decentralized design there is no global entity to coordinate the resource allocation. A local controller resides on each node and decides the local allocation with no, or limited knowledge of the state of the other nodes from the system. An application manager negotiates with the local controllers the resource allocation for its application [69]. In this case, another problem that needs to be considered is the coordination between the application managers.

4.4 Summary

In this section we presented the mechanisms of resource provisioning that used virtualization as a building block. We classified these mechanisms in three types, based on what type of resource provisioning abstraction they provide to users: i) static dedicated virtual environments; ii)dynamic shared virtual environments; iii) dynamic dedicated virtual environments. Static dedicated virtual environments can be too rigid when application requirements or the underlying environment change in time. Thus, they provide a generic abstraction for different usage scenarios but they cannot provide higher level guarantees for users. Dynamic shared virtual environments can be used to ensure a fair share of resources between different groups of applications or groups of users, while delegating resource provisioning for individual applications to other scheduling layers. Thus, they provide a lower-level coarse-grain resource provisioning abstraction, not suitable for users themselves, but more for infrastructure administrators. Dynamic dedicated virtual environments can be used to adapt the resource allocation to application's demand. They provide better flexibility than the two previous mechanisms. However, they tend to be specific to the types of applications that are running on the same infrastructure and efforts still need to be made to simultaneusly address multiple constraints imposed by users.

Project Name	**Goal**	**Resource Management Scope**	**App Runtime Management**	**VM Resource Allocation**	**VM Dynamic Placement**	**System Resources**	**Architecture Model**
Friendly virtual machines [74]	Self-adaptation of the virtual machines to the current resource availability	single node	Heuristics	Coupled with App runtime management	No	CPU, Memory	decentralized
Active coordination - toward effectively managing virtualized multicore clouds [29]	Maximize resource utilization by adapting resource allocation and support different classes of service	single node	Performance monitoring	Heuristics	Simulate by suspending the VM	CPU, network	2 level of control
Autonomic live adaptation of virtualized environments [55]	Maximize resource utilization and application performance	multi-domain	Resource usage monitoring	Heuristics	Coupled with resource allocation	CPU, memory	Centralized
Vconf [53]	Maximize resource utilization and minimize probability of breaching SLAs	single node	Monitoring	Machine-learning algorithm	No	CPU, memory	Centralized
Enforcing SLAs in Scientific Clouds [44]	Meet deadlines for scientific applications in clouds	single domain	Fuzzy-based algorithm	No; Cloud provider-dependent	No; Cloud provider-dependent	VM	Decentralized
Autonomic resource management in virtualized datacenters using fuzzy logic-based approaches [72]	Maximize resource utilization and minimize probability of breaching SLAs	single domain	Fuzzy-based algorithm	greedy algorithm	No	CPU	2 level control

Adaptive control of virtualized resources in utility computing environments [48]	Provide response time guarantees in consolidated servers	single domain	Feedback integral controller	Policies and integral controllers	No	CPU	2 level control
Self-tunning virtual machines [49, 50]	Run best-effort applications in parallel with deadline-driven applications	single domain	Performance monitoring	Feedback integral controller	No	CPU	2 level control
Resource allocation for datacenters using analytic performance models [5]	Minimize the probability of breaching SLAs in a datacenter	single domain	utility functions	Beam-seach algorithm	No	PM	2 level control
SLA-aware virtual resource management in cloud infrastructure [43]	Application SLA-aware consolidation in datacenters	single domain	weighted utility functions	CSP	CSP	VM	2 level control
Rhizoma [73]	Optimize application deployment in a shared infrastructure	single domain	Utility functions	CSP	Coupled with resource allocation	PM	Decentralized
AppRAISE [69]	Application performance management in datacenters	single domain	feedback and feedforward controllers	adaptive controllers and policy-based local resource arbitration	No	CPU	decentralized
1000 islands [76]	Automate capacity and workload management in datacenters	single domain	feedback controller	policies based on priority levels	simulated annealing algorithm	CPU	multi-level

Table 2: Summary of dynamic resource provisioning solutions.

5 Conclusions

This report presented how virtualization was used in managing distributed shared infrastructures. We have argued that is difficult to share resources of distributed computational infrastructures between different applications in a way to meet the different guarantees that users could impose. As traditional resource provisioning mechanisms still do not provide enough flexibility for solving this problem, we have focused on more recent management mechanisms, based on virtualization technologies.

Virtualization became popular as a mechanism for sharing distributed infrastructures between multiple applications. A lot of recent work focused on developing solutions to provide users with virtual environments. Different research groups addressed particular aspects of the management of virtual machines. Such aspects included configuring virtual environments, minimizing the time required for their instantiation, or hiding the overhead of managing them from the user. These aspects were addressed at a single domain level (i.e. datacenter or single resource provider) and for multiple administrative domains. However, this work did not properly addressed the problem of efficiently managing resources for virtual machines. Resource provisioning was either done on an immediate or best-effort basis, either was based on existing job-based systems.

To address this gap, different resource provisioning mechanisms for virtual environments were developed. We classified these mechanisms in two types: static and dynamic. Static provisioning mechanisms were introduced to accomodate different usage scenarios that could appear on a shared infrastructure. However, because application resource demands and resource availability can change in time, these mechanisms are not enough to provide an acceptable level of QoS to users.

Dynamic resource provisioning mechanisms were developed to provide virtual clusters with dynamic capacity and to manage applications in consolidated datacenters, They take advantage of virtualization to adjust the capacity of the virtual cluster to current workloads. However, even if these mechanisms address the case of maximizing resource utlization, they don't address the case of supporting different user requirements or application demands on the same physical infrastructure. Lastly, dynamic resource provisioning mechanisms were designed in datacenters to maximize resource utilization and reduce power consumption, while meeting specific user contraints, usually expressed in response time or throughput. Such mechanisms packing multiple virtual machines on the same physical servers while monitoring the performance of applications running in them and adjusting their resource allocation accordingly. However, most of these mechanisms were developed specifically for the case of web applications hosted in datacenters, and they were less considered for ensuring other types of user guarantees, like deadlines for scientific applications.

In this context, we can conclude that too few efforts have been made towards taking full advantage of virtualization capabilities. More scalable mechanisms need to be designed such they are capable of supporting different types of guarantees and different types of applications on the same infrastructure.

References

[1] Sumalatha Adabala, Vineet Chadha, Puneet Chawla, Renato Figueiredo, José Fortes, Ivan Krsul, Andrea Matsunaga, Mauricio Tsugawa, Jian Zhang, Ming Zhao, Liping Zhu, and Xiaomin Zhu. From virtualized resources to virtual computing grids: the in-vigo system. In *Future Gener. Comput. Syst.*, 2005.

[2] AmazonEC2. http://aws.amazon.com/ec2/.

[3] 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 *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP '03)*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[4] Andy Bavier, Mic Bowman, Brent Chun, David Culler, Scott Karlin, Steve Muir, Larry Peterson, Timothy Roscoe, Tammo Spalink, and Mike Wawrzoniak. Operating system support for planetary-scale network services. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 19–19, Berkeley, CA, USA, 2004. USENIX Association.

[5] M.N. Bennani and D.A. Menasce. Resource allocation for autonomic data centers using analytic performance models. pages 229 –240, 2005.

[6] R. Bradshaw, N. Desai, T. Freeman, and K. Keahey. Proceedings of the terragrid conference. In *A Scalable Approach to Deploying and Managing Applications*, 2007.

[7] Jeffrey S. Chase, David E. Irwin, Laura E. Grit, Justin D. Moore, and Sara E. Sprenkle. Dynamic virtual clusters in a grid site manager. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, 2003.

[8] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[9] W. Emeneker, D. Stanzione, and H.P.C. Initiative. Increasing reliability through dynamic virtual clustering. In *High Availability and Performance Computing Workshop*, 2006.

[10] Wesley Emeneker and Dan Stanzione. Dynamic virtual clustering. In *CLUSTER '07: Proceedings of the 2007 IEEE International Conference on Cluster Computing*, pages 84–90, Washington, DC, USA, 2007. IEEE Computer Society.

[11] Christian Engelmann, Stephen Scott, Hong Ong, Geoffroy Vallee, and Thomas Naughton. Configurable virtualized system environments for high performance computing. In *In Proceedings of the 1 st Workshop on System-level Virtualization for High Performance Computing (HPCVirt)*, 2007.

[12] Sun Grid Engine. http://www.sun.com/software/sge.

[13] Hans-Joachim Fallenbeck, Niels Picht, Matthew Smith, and Bernd Freisleben. Xen and the art of cluster scheduling. In *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, pages 237–244. IEEE Press, 2006.

[14] J. Renato Figueiredo, Peter A. Dinda, and Jose A.B. Fortes. A case for grid computing on virtual machines. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, 2003.

[15] I. Foster, T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, and X. Zhang. Virtual clusters for grid communities. In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, 2006.

[16] Timothy Freeman and Katarzyna Keahey. Flying low: Simple leases with workspace pilot. In *Proceedings of the 14th international Euro-Par conference on Parallel Processing*, pages 499–509, Berlin, Heidelberg, 2008.

[17] Yun Fu, Jeffrey Chase, Brent Chun, Stephen Schwab, and Amin Vahdat. Sharp: an architecture for secure resource peering. In *In Proceedings of the 19th ACM Symposium on Operating System Principles*, pages 133–148, New York, NY, USA, 2003. ACM.

[18] Jerome Gallard, Christine Morin, and Adrien Lebre. Saline: Improving best-effort job management in grids. In *Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2010.

[19] A. Gavrilovska, S. Kumar, H. Raj, K. Schwan, V. Gupta, R. Nathuji, R. Niranjan, A. Ranadive, and P. Saraiya. High-performance hypervisor architectures: Virtualization in hpc systems. In *1st Workshop on System-level Virtualization for High Performance Computing*, 2007.

[20] L. Grit, D. Irwin, V. Marupadi, P. Shivam, A. Yumerefendi, J. Chase, and J. Albrecht. Harnessing virtual machine resource control for job management. In *Proceedings of the First International Workshop on Virtualization Technology in Distributed Computing*, 2006.

[21] Laura Grit, David Irwin, Aydan Yumerefendi, and Jeff Chase. Virtual machine hosting for networked clusters: Building the foundations for "autonomic" orchestration. In *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, page 7, Washington, DC, USA, 2006. IEEE Computer Society.

[22] Fabien Hermenier, Xavier Lorca, and Jean-Marc Menaud. Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual Execution Environments*, 2009.

[23] Mike Hibler, Robert Ricci, Leigh Stoller, Jonathon Duerig, Shashi Guruprasad, Tim Stack, Kirk Webb, and Jay Lepreau. Large-scale virtualization in the emulab network testbed. In *In Proceedings of the 2008 USENIX Annual Technical Conference*, pages 113–128, Boston, MA, 2008.

[24] Takahiro Hirofuchi, Takeshi Yokoi, Tadashi Ebara, Yusuke Tanimura, Hirotaka Ogawa, Hidetomo Nakada, Yoshio Tanaka, and Satoshi Sekiguchi. A multi-site virtual cluster system for wide area networks. In *First USENIX Workshop on Large-Scale Computing*, pages 1–10, Berkeley, CA, USA, 2008. USENIX Association.

[25] Wei Huang, Jiuxing Liu, Bulent Abali, and K. Dhabaleswar Panda. A case for high performance computing with virtual machines. In *Proceedings of the 20th annual international conference on Supercomputing*, 2006.

[26] David Irwin, Jeffrey Chase, Laura Grit, Aydan Yumerefendi, David Becker, and Kenneth G. Yocum. Sharing networked resources with brokered leases. In *Proceedings of the 2006 USENIX Annual Technical Conference*, pages 18–18, Berkeley, CA, USA, 2006. USENIX Association.

[27] Ardalan Kangarlou, Patrick Eugster, and Dongyan Xu. Vnsnap: Taking snapshots of virtual networked environments with minimal downtime. In *Proceedings of the 2009 IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 524–533, 2009.

[28] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa. Science clouds: Early experiences in cloud computing for scientific applications. *Cloud Computing and Applications*, 2008, 2008.

[29] M Kesavan, A Ranadive, A Gavrilovska, and K Schwan. Active coordination (act) - toward effectively managing virtualized multicore clouds. In *In Proceedings of the 2008 IEEE International Conference on Cluster Computing*, pages 23–32, Tsukuba, 2008.

[30] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the Linux virtual machine monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, 2007.

[31] Nadir Kiyanclar, A. Gregory Koenig, and William Yurcik. Maestro-vc: On-demand secure cluster computing using virtualization. In *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid Workshops*, 2006.

[32] Younggyun Koh, R. Knauerhase, P. Brett, M. Bowman, Zhihua Wen, and C. Pu. An analysis of performance interference effects in virtual environments. In *2007 IEEE International Symposium on Performance Analysis of Systems Software*, pages 200 –209, 2007.

[33] Ivan Krsul, Arijit Ganguly, Jian Zhang, Jose A. B. Fortes, and Renato J. Figueiredo. Vmplants: Providing and managing virtual machine execution environments for grid computing. In *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, 2004.

[34] H.A. Lagar-Cavilla, J.A. Whitney, A.M. Scannell, P. Patchin, S.M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan. Snowflock: Rapid virtual machine cloning for cloud computing. In *Proceedings of the Fourth ACM European Conference on Computer Systems*, 2009.

[35] LXC. http://lxc.sourceforge.net/.

[36] Martin W. Margo, Kenneth Yoshimoto, Patricia Kovatch, and Phil Andrews. Impact of reservations on production job scheduling. In *JSSPP'07: Proceedings of the 13th international conference on Job scheduling strategies for parallel processing*, pages 116–131, Berlin, Heidelberg, 2008. Springer-Verlag.

[37] Maui. http://www.nsc.liu.se/systems/retiredsystems/grendel/maui.html.

[38] Marvin McNett, Diwaker Gupta, Amin Vahdat, and Geoffrey M. Voelker. Usher: an extensible framework for managing custers of virtual machines. In *LISA'07: Proceedings of the 21st conference on Large Installation System Administration Conference*, pages 1–15, Berkeley, CA, USA, 2007. USENIX Association.

[39] Moab. http://www.clusterresources.com/products/moab-cluster-suite.php.

[40] Michael A. Murphy, Michael Fenn, and Sebastien Goasguen. Virtual organization clusters. In *PDP '09: Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pages 401–408, Washington, DC, USA, 2009. IEEE Computer Society.

[41] Michael A. Murphy, Brandon Kagey, Michael Fenn, and Sebastien Goasguen. Dynamic provisioning of virtual organization clusters. In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 364–371, Washington, DC, USA, 2009. IEEE Computer Society.

[42] Arun Babu Nagarajan, Frank Mueller, Christian Engelmann, and Stephen L. Scott. Proactive fault tolerance for hpc with xen virtualization. In *ICS '07: Proceedings of the 21st annual international conference on Supercomputing*, pages 23–32, New York, NY, USA, 2007. ACM.

[43] Hien Nguyen Van, Frederic Dang Tran, and Jean-Marc Menaud. SLA-aware virtual resource management for cloud infrastructures. In *9th IEEE International Conference on Computer and Information Technology (CIT'09) 9th IEEE International Conference on Computer and Information Technology (CIT'09)*, pages 1–8, 2009.

[44] Oliver Niehrster, Andre Brinkmann, Gregor Fels, Jens Kruger, and Jens Simon. Enforcing slas in scientific clouds. In *In Proceedings of the 2010 IEEE International Conference on Cluster Computing*, 2010.

[45] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009.

[46] Hong Ong, Natthapol Saragol, Kasidit Chanchio, and Chokchai Leangsuksun. Vccp: A transparent, coordinated checkpointing system for virtualization-based cluster computing. In *Proceedings of the 2009 IEEE International Conference on Cluster Computing*, pages 1–10, 2009.

[47] OpenVZ. http://wiki.openvz.org.

[48] Pradeep Padala, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. Adaptive control of virtualized resources in utility computing environments. *SIGOPS Oper. Syst. Rev.*, 41(3):289–302, 2007.

[49] Sang-Min Park and Marty Humphrey. Feedback-controlled resource sharing for predictable escience. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–11, Piscataway, NJ, USA, 2008. IEEE Press.

[50] Sang-Min Park and Marty Humphrey. Self-tuning virtual machines for predictable escience. In *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 356–363, Washington, DC, USA, 2009. IEEE Computer Society.

[51] Lavanya Ramakrishnan, David Irwin, Laura Grit, Aydan Yumerefendi, Adriana Iamnitchi, and Jeff Chase. Toward a doctrine of containment: grid hosting with adaptive resource control. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 101, New York, NY, USA, 2006. ACM.

[52] Adit Ranadive, Mukil Kesavan, Ada Gavrilovska, and Karsten Schwan. Performance implications of virtualizing multicore cluster machines. In *HPCVirt '08: Proceedings of the 2nd workshop on System-level virtualization for high performance computing*, pages 1–8, New York, NY, USA, 2008.

[53] Jia Rao, Xiangping Bu, Cheng-Zhong Xu, Leyi Wang, and George Yin. Vconf: a reinforcement learning approach to virtual machines auto-configuration. In *ICAC '09: Proceedings of the 6th international conference on Autonomic computing*, pages 137–146, New York, NY, USA, 2009. ACM.

[54] Paul Ruth, Phil McGachey, and Dongyan Xu. Viocluster: virtualization for dynamic computational domains. In *Proceedings of Cluster 2005: IEEE International Conference on Cluster Computing*, 2005.

[55] Paul Ruth, Junghwan Rhee, Dongyan Xu, Rick Kennell, and Sebastien Goasguen. Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure. In *IEEE International Conference on Autonomic Computing*, 2006.

[56] Alex Shoykhet, Jack Lange, and Peter A. Dinda. Virtuoso: A system for virtual machine marketplaces. Technical report, Computer Science Department, Northwest University, July 2004.

[57] B. Sotomayor, R.S. Montero, I.M. Llorente, and I. Foster. An Open Source Solution for Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing*, 13(5):14–22, 2009.

[58] Borja Sotomayor, Kate Keahey, and Ian Foster. Overhead matters: A model for virtual resource management. In *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, 2006.

[59] Borja Sotomayor, Kate Keahey, and Ian Foster. Combining batch execution and leasing using virtual machines. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing*, 2008.

[60] Borja Sotomayor, Kate Keahey, Ian Foster, and Tim Freeman. Enabling cost-effective resource leases with virtual machines. In *Hot Topics session in ACM/IEEE International Symposium on High Performance Distributed Computing 2007*, Monterey Bay, CA (USA), 2007.

[61] Borja Sotomayor, Rubén Santiago Montero, Ignacio Martín Llorente, and Ian Foster. Resource leasing and the art of suspending virtual machines. In *Proceedings of the 2009 11th IEEE International Conference on High Performance Computing and Communications*, pages 59–68, Washington, DC, USA, 2009. IEEE Computer Society.

[62] I. Ananh Sundararaj, Ashish Gupta, and Peter A. Dinda. Increasing application performance in virtual environments through run-time inference and adaptation. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing*, 2005.

[63] Gerald Tesauro, Jeffrey O. Kephart, and Rajarshi Das. Utility functions in autonomic systems. In *ICAC '04: Proceedings of the First International Conference on Autonomic Computing*, pages 70–77, Washington, DC, USA, 2004. IEEE Computer Society.

[64] G. Vallee, T. Naughton, C. Engelmann, H. Ong, and S.L. Scott. System-level virtualization for high performance computing. In *Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, 2008.

[65] G. Vallée and S. Scott. Xen-oscar for cluster virtualization. In *Frontiers of High Performance Computing and Networking–ISPA 2006 Workshops*, pages 487–498. Springer, 2006.

[66] VirtualBox. http://www.virtualbox.org/.

[67] Linux VServer. http://linux-vserver.org/.

[68] John Paul Walters and Vipin Chaudhary. A fault-tolerant strategy for virtualized hpc clusters. *J. Supercomputing*, 50(3):209–239, 2009.

[69] Zhikui Wang, Yuan Chen, D. Gmach, S. Singhal, B.J. Watson, W. Rivera, Xiaoyun Zhu, and C.D. Hyser. Appraise: application-level performance management in virtualized server environments. *IEEE Transactions on Network and Service Management*, 6(4):240 –254, 2009.

[70] WMware. http://www.wmware.com.

[71] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation*, pages 229–242, 2007.

[72] Jing Xu, Ming Zhao, José Fortes, Robert Carpenter, and Mazin Yousif. Autonomic resource management in virtualized data centers using fuzzy logic-based approaches. *Cluster Computing*, 11(3):213–227, 2008.

[73] Qin Yin, Adrian Schüpbach, Justin Cappos, Andrew Baumann, and Timothy Roscoe. Rhizoma: a runtime for self-deploying, self-managing overlays. In *Middleware '09: Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, pages 1–20, New York, NY, USA, 2009. Springer-Verlag New York, Inc.

[74] Yuting Zhang, Azer Bestavros, Mina Guirguis, Ibrahim Matta, and Richard West. Friendly virtual machines: leveraging a feedback-control model for application

adaptation. In *VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, pages 2–12, New York, NY, USA, 2005. ACM.

[75] Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, Pradeep Padala, and Kang Shin. What does control theory bring to systems research? *SIGOPS Oper. Syst. Rev.*, 43(1):62–69, 2009.

[76] Xiaoyun Zhu, Donald Young, Brian J. Watson, Zhikui Wang, Jerry Rolia, Sharad Singhal, Bret Mckee, Chris Hyser, Daniel Gmach, Robert Gardner, Tom Christian, and Ludmila Cherkasova. 1000 islands: an integrated approach to resource management for virtualized data centers. *Cluster Computing*, 12(1):45–57, 2009.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803