INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

VMdeploy: Improving Best-Effort Job Management in Grid'5000

Jérôme Gallard — Adrien Lèbre — Oana Goga — Christine Morin

N° 6764

Décembre 2008

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--6764--FR+ENG

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE | INRIA | centre de recherche RENNES - BRETAGNE ATLANTIQUE

VMdeploy: Improving Best-Effort Job Management in Grid'5000 *

Jérôme Gallard† , Adrien Lèbre‡ , Oana Goga† , Christine Morin†

Thème NUM — Systèmes numériques
Équipe-Projet PARIS

Rapport de recherche n° 6764 — Décembre 2008 — 16 pages

Abstract: Virtualization technologies have recently gained a lot of interest in Grid computing as they allow flexible resource management. Grid'5000 (G5K) is a French national Grid platform used for computer science research to experiment all layers of Grid software. Computer scientists reserve G5K nodes prior to their experiments. In G5K some low priority jobs are executed in best effort mode on the node idle time slots when the latter are not part of any reservation. However, best-effort jobs may be killed at any time by the Grid job scheduler when the nodes they use are subject to higher priority reservation. This behaviour leads potentially to a huge waste of compute time or at least requires users to deal with checkpoints of their best efforts jobs.

In this paper, we describe the design and implementation of the `VMdeploy` framework which exploits virtual machines for executing best effort jobs in order to solve the best-effort issue in G5K platform. `VMdeploy` manages snapshots of the best effort jobs transparently to their users and thus ensures the progress of these jobs avoiding most of the waste of resources. Results of a preliminary experimental evaluation are presented. While designed in the context of G5K, `VMdeploy` can be used in combination of any job scheduler in clusters and grids.

Key-words: Virtualization, Grid, Best Effort jobs, Scheduling, Resource Management.

* The INRIA team carries out this research work in the framework of the XtreemOS project partially funded by the European Commission under contract #FP6-033576.

† INRIA Rennes - Bretagne Atlantique, Rennes France – firstname.lastname@irisa.fr
‡ EMN, France – adrien.lebre@emn.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

VMdeploy: Comment améliorer la gestion des travaux de type "best-effort" dans Grid'5000.

Résumé : Les technologies de virtualisation ont récemment eu un gain d'intérêt dans le domaine de la Grille et cela est dû principalement au fait qu'elles permettent une plus grande flexibilité dans la gestion des ressources. Grid'5000 (G5K) est une Grille nationale Française utilisée pour des expérimentations scientifiques à grande échelle. Pour pouvoir réaliser leurs expérimentations, les utilisateurs de G5K doivent réserver leurs nœuds. Dans G5K, des travaux de faible priorité ("best-effort" – faire au mieux) sont exécutés sur des nœuds disponibles, c'est-à-dire, ne faisant parti d'aucune réservation. Ces travaux de type "best-effort" peuvent être retirés des nœuds à tout moment par l'ordonnanceur de la Grille quand des travaux de priorité supérieure sont soumis. Ce comportement emmène potentiellement à des pertes de temps de calcul, ou, contraint les utilisateurs à mettre en place des méthodes de sauvegarde/restauration de point de reprise de leurs travaux "best-effort".

Dans ce document, nous décrivons l'architecture ainsi que l'implémentation de `VMdeploy`, notre prototype. `VMdeploy` exploite les fonctionnalités des machines virtuelles (VM) pour exécuter des travaux de type "best-effort" afin d'optimiser leur gestion. `VMdeploy` gère la création ainsi que le déplacement et la suspension/redémarrage des VM de manière transparente pour les utilisateurs afin de réduire au mieux la perte de temps de calcul. Les premières expérimentations que nous présentons se montrent concluantes. Enfin, bien qu'il ait été conçu dans le contexte de G5K, `VMdeploy` peut être utilisé avec n'importe quel ordonnanceur de grappe ou grille.

Mots-clés : Virtualisation, Grille, Travaux de type "best-effort", Ordonnancement, Gestion des ressources.

1 Introduction

Clusters and grids are used for a wide range of applications providing high-performance computing, large storage capacity, and high throughput communication. The most common way of exploiting this kind of distributed architectures consists in using dedicated services in particular batch schedulers in order to get resources at (i) a particular time ("reservation"), (ii) as soon as possible ("interactive") or (iii) when the resources are idle ("best-effort").

Several works have been extended to provide more flexibility to users (deployment of dedicated environments [26, 5], lease concept [23], ...). However, cluster and grid usage is still based on a reservation scheme where a "static" set of resources is assigned to a "job" (or a "user") during a bounded amount of time.

This model of using clusters or grids leads to a coarse-grain exploitation of the architecture since resources are simply reassigned to another job/user at the end of the slot without considering the real completion of applications. In the best case, the time-slot is larger and resources are simply under used. In the worst case, running applications are withdrawn from their resources leading potentially to the loss of all the performed calculations and requiring to execute once again the same request. In other words, the set of resources assigned to a job/user cannot evolve in time according to (i) application needs and (ii) cluster or grid resource changes.

In this paper, we focus on this lack of "dynamicity" in cluster or grid use. The different modes for exploiting clusters or grids (i.e. reservation, interactive or best-effort) imply several issues to consider. As a consequence, we voluntary chose to firstly address the case where applications are fatally taken out from resources.present contribution.

Usually exploited for fault-tolerance issues, checkpointing solutions, like checkpointing-based resource preemption, have been proposed to partially improve dynamicity in clusters. However, these methods are strongly middleware or OS dependant [11, 25]: they require either to link applications with dedicated checkpointing libraries or to exploit a checkpointing capable OS. Thus, moreover requiring specialized software stacks, these methods are not appropriate for a heterogeneous environment such as a grid (checkpointing dependencies, both hardware and software, limiting the locations where the application can be restarted).

Thanks to the latest improvements, virtualization tools could solve such issues and thus tackle the challenge addressed in this paper: improving "transparent dynamicity" in grid usage. Virtual machines (VMs) become more and more popular [20] in the context of Grid and more recently Cloud Computing, providing flexible, isolated and powerful execution environments. By specifically using VM capabilities such as snapshot, migrate, suspend and resume, it becomes possible to significantly reduce the loss of computation time and to provide a more dynamic usage of distributed architectures such as clusters and grids.

In this paper, we analyze what are the main challenges in designing and implementing a generic framework exploiting snapshotting and migration VM capabilities in coordination with any batch scheduler in a grid context. This framework aims at providing a more "dynamic" use of distributed architectures and so, at significantly reducing loss of computation time and thus power consumption. Keeping in mind each constraint that we have to take into account, we present a first prototype addressing the best-effort issue in the Grid'5000 architecture [5]. Thanks to this prototype, entitled `VMdeploy`, best-effort jobs can be transparently submitted in VMs, so that the computation can be dynamically migrated or suspended and then resumed each time the resources are taken away from the users. Such an approach results in better performance concerning the total execution time of best-effort requests and a large benefit according to power consumption. Our framework is able to automatically take in consideration best-effort requests, to deploy them into VMs and monitor them until their completion.

Advanced features such as dynamical load-balancing is out-of-scope of this paper. The aim of this work is to address the different challenges that the community has to tackle in order to provide an appropriate framework solving the "dynamicity" criteria in cluster and grid usage.

The remainder of the paper is organized as follow: Section 2 motivates our work by addressing the impact of a coarse-grain management of the Grid'5000 best-effort jobs and the benefits of the latest VM capabilities in such a context. Section 3 is devoted to the challenges of each mandatory step to schedule and monitor a VM job at cluster and grid level. Section 4 describes the implemented prototype and discusses experiments. Related work is addressed in Section 5. Finally, Section 6 concludes and gives several perspectives.

2 Motivations

The main objective of this work consists in providing a way to improve dynamicity in the manner of using clusters and grids to finer exploit them. After emphasizing how coarse-grain management of best-effort jobs can lead to an important loss of computation (and power consumption) this section focuses on the benefits of using VM technologies to reduce these issues.

2.1 Best-effort and Current Batch Schedulers

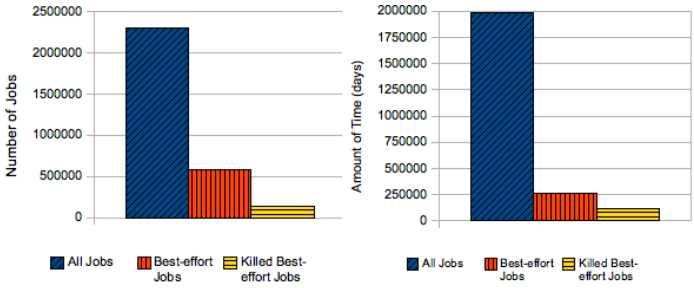
The "best-effort" concept has been suggested in the batch scheduler context in order to back-fill clusters or grids during free slots which do not fit well time-bounded requests. This particular mode does not provide any guarantees and jobs can be cancelled before the end of their allowed time. This leads to utilization problems since the jobs are simply withdrawn from their resources without considering if the performed calculations are lost. As the next section shows, this behavior becomes more and more significant as the number of best effort jobs increases in a large cluster or in a grid such as G5K.

2.1.1 Current Status in Grid'5000

G5K aims at building a highly reconfigurable, controllable and monitorable experimental Grid platform gathering 9 sites geographically distributed in France featuring a total of 5000 processors. This is a heterogeneous architecture exploited by multiple users and several kinds of applications (business, scientific). The batch scheduler used in G5K is OAR [6]. It provides an extension, entitled "oargrid", to manage grid resources as it does at cluster level. Last but not the least, G5K allows the deployment of customized environments on the grid nodes thanks to the `Kadeploy` software [5]. `Kadeploy` enables to deploy any user environment directly on the bare hardware. Unfortunately, it does not provide any feature to suspend or restart the physical images.

To show the importance of taking into consideration the best-effort issue, we gathered and made some statistics on the OAR traces available from the Grid Workloads Archive [13]. The traces were gathered from September 2005 (beginning of G5K initiative) to August 2008. Figure 2.1.1 presents the results. As we assumed, best-effort jobs are quite used on G5K (25% of the jobs) and 22% of them are simply withdrawn by the OAR scheduler leading to more than 100000 days wasted.

To avoid data loss, users of best-effort jobs have to setup complex systems to either periodically save results or to dynamically resubmit once again the lost jobs. The available solutions based on checkpointing methods are often not appropriated to the grid and/or to the applications. Finally, developing a customized framework for each application is a difficult and tedious task and obviously not transparent. As virtual machine supervisors offer migration, suspend/resume



since september 2005 a) Distribution of the Grid'5000 jobs

25% of best-effort jobs representing more than 260000 days.

	Best-effort Jobs		**Killed Best-efforts Jobs**		
	Nb	**days**	**Nb**	**days**	**Time %**
Total	578438	**262907**	132504	**113353**	**43.12%**

b) Best-effort details: more than 40% of the CPU time is wasted ("Nb" - Number of requests, "days"- corresponding global amount of CPU time in days)

Figure 1: Best-effort usage in Grid'5000

and snapshotting capabilities, we can exploit them to correct this particular weakness by running such jobs inside VMs.

2.2 Benefits of Virtualization Technologies

Virtualization is an active research subject since the 70's. The recently increased world-wide interest of researchers, developers and enterprise businesses in virtualization sparked a few years ago with the development of lightweight hypervisor technologies [2, 3]. Over the last years the technology has matured up to the point that processor manufacturers, like Intel and AMD, incorporated hardware virtualization support in their products.

VM technologies provide flexible and powerful execution environments, offering isolation and security mechanisms, customization and encapsulation of entire application environments. Moreover, they allow the bare hardware to be strongly decoupled from the system software which is a predominant feature in the context of distributed architectures such as grids. Grids are composed of several resources which can be heterogeneous, geographically distributed and which can potentially belong to distinct administrative areas. In addition, grids are exploited concurrently by multiple users who want to execute their applications in a secure and efficient way. In this particular context, VMs can be exploited to encapsulate jobs and then make grid resource management easier. The challenge of managing applications on grids is moved to the problem of managing VMs on grids. Thanks to VM capabilities, developing a framework to improve dynamicity in grid usage could be reached: VMs can be either suspended and resumed or migrated from one physical architecture to another at any time.

The next section addresses the different challenges to tackle in order to design and implement such a VM management framework at grid level.

3 Designing a VM Management Framework

The goal of our framework consists of exploiting snapshotting and migration VM capabilities on top of traditional batch scheduler or reservation services in order to provide a more "dynamic" usage of distributed architectures.

Designing and implementing such a framework require to consider each step of a common job submission in a grid context : (i) VM image creation (initial setup), (ii) VM image repository (central/distributed, clusterwide/gridwide) (iii) job submission, VM deployment and job starting, (iiii) VM/job life cycle (live migration, suspend/resume, periodical snapshots, ...) and finally (iiiii) job completion and VM shutdown. Based on this scenario, we outline three major points that we clearly identified as challenges: (i) VM storage management, (ii) network configuration and mobility and (iii) real-time VM management.

3.1 VM Storage Management

3.1.1 Image Creation

Our framework should be able to select the right VM image according to the user needs. Two cases have to be considered : (i) the submitted job can be executed on top of a standard environment (e.g. Gnu/Linux) or (ii) it requires a particular environment. If the first case can be easily solved by using a standard VM, the second one implies the creation of a particular image. From our point of view, the creation itself must be done externally by using an advanced mechanism such as a webservice application. However and due to the large number of environment creation utilities, it is important to be able to convert a particular environment into a VM image format understandable by our framework.

3.1.2 Initial Deployment

The deployment of a VM can be divided in two distinct phases : (i) deployment of an appropriated hypervisor on target nodes and (ii) deployment of the VM. Like the image creation, efficient deployment of the hypervisor system itself (the host OS) is beyond the scope of the paper. The proposal should provide a generic layer in order to request hypervisor deployments when it is required. However, the current trends in the use of VM in grids let us to think that each node will provide an hypervisor (leading to simply skip this step).

The efficient VM deployment issue can be compared to previous hardware deployment challenge: providing and setting up an OS from one node to several others. Three solutions can be considered :

- *Diskless approach*, it has been for a while one of the famous approach to set up particular operating systems on several nodes. It consists in using a distributed file system (from the well-known NFS server to the most recent distributed file systems such as Lustre) as a repository for all images. In our particular case, each node can launch a particular VM image by mounting the distributed file system. This approach implies consistency issues between several nodes executing the same VM (and thus saving their changes on the same file hierarchy physically stored on the distributed file system). Exploiting a separate hierarchy for each node executing the same VM leads quickly to space storage problems. A better solution consists in exploiting Copy-On-Write techniques [4] where each VM saves its own modifications in a particular file. This *diff* file can be written either locally or on the distributed file system. Unfortunately, diskless approaches strongly depends on distributed file system scalability and limitations. For example, only few distributed file systems can be efficiently exploited at grid level.

- *Local VM image*, this approach consists in deploying once for all the VM images on each node (exploiting the large local storage space available). When a job requires a particular VM, the hypervisor can simply access the local image instead of a remote one. However propagating the changes of each VM to each node could become really complex. Moreover

it can be expensive (even with copy-on-write approaches). In addition, this approach can lead to security issues since we should ensure that no users can access hard drives in order to corrupt VM images.

- *Efficient VM image copy from a dedicated repository to the target node(s)* The idea consists in exploiting advanced tools in order to dynamically deploy the VM image from a central repository to the target node(s). A central repository is used to store the VM environments. VMs are dynamically distributed to the target nodes when it is required thanks to an advanced copy mechanisms.

Due to the simplicity of the latter solution with regard to the former one, initial deployment in our proposal is based on an efficient copy approach. It exploits the `TakTuk` utility [24] that provides a dynamic deployment using a binomial tree.

3.1.3 Management of VM Snapshots

Snapshotting VM periodically leads to another storage challenge. For each running VM, we have to consider two contents : (i) the current image, that is the reference image initially deployed plus the modifications applied since the beginning of the execution (the *diff* file formerly introduced) and (ii) the current volatile state, that is the file representing the memory. These two contents should be periodically saved for each VM running in the cluster or the grid. Retrieving the current image of suspended VMs is a particular case of the snapshot management where the memory state has been already serialized in a particular file. As a consequence, this section does not directly address this issue.

Considering that reference VM images are available on a central repository, using a Copy-on-Write strategy in order to store VM image modifications on each node would really improve efficiency. Thanks to it, the snapshot process can be reduced to periodically save the *diff* file and the memory state. Nevertheless and even if the amount of data per VM is less significant than a complete copy, saving these states in an efficient manner is still a challenge and requires to analyze several possibilities:

- Using a distributed file system, as we previously described, this approach depends on distributed file system scalability and grid limitations.
- Saving snapshot locally seems to be an interesting approach at the first sight. Snapshot data is stored locally and when another job with a higher priority preempts the node, the VM can be simply killed or suspended. It can be restarted (i) when the new job completes or (ii) when new resources are freed in the grid. In that particular case the VM state has to be copied to the free node and restarted. Unfortunately such an approach can significantly impact the performance of the local node since the new job competes with the copy operation that intensively uses hard drive and network card resources. Moreover in the event of a node crash, data is not reachable. So, saving snasphot locally would require an independent and reachable hard drive similarly as current management network cards available in recent servers.
- Copy from the target node(s) to a dedicated repository leads unfortunately to important bottlenecks since a unique job can imply hundred of VMs.

Saving snapshots in an efficient way is a real challenge and is still under investigation. Currently, our proposal copies each snapshot to a dedicated repository. Each copy overwrites the latest one.

3.2 Network Configuration and Mobility

From the network point of view, the challenges depend on the VM network management provided by each hypervisor. The issues could be divided in two points: (i) configuring MAC and IP addresses gridwide and (ii) take into account VM migration from one site to another one.

3.2.1 VM network card configuration

During the VM creation process (when the reference image has been built), each environment has been configured with specific network parameters for both MAC and IP addresses. As a consequence, we have to ensure that the current network configuration of a VM is not going to disturb physical infrastructure or other VM network configuration before starting it. The challenge consists in dynamically assigning a new MAC and/or a new IP when it is needed. In order to be able to dynamically reassign a MAC and/or a new IP, we have to set up particular mechanisms :

- *considering a central DHCP server:* in this particular case, VMs send a DHCP request and receive their complete network configuration as physical nodes do in a cluster. In addition to deal with the scalability issue, designing and implementing a Grid DHPC server seems to be technically not possible. Firstly, each site of a grid belongs potentially to a distinct network class and usually exploits its own DHCP server. Secondly, the VM MAC assignment is based on the address of each hypervisor and it is not possible to guarantee that two VMs on two distinct nodes do not receive the same MAC addresses.
- *deploying a hybrid DHCP server per job:* this approach consists in exploiting one more node for each reservation (ie, if the user requests for 100 VMs deployed on 100 physical nodes, then, the system makes a reservation for 101 nodes). This node is in charge of assigning a particular ID to all virtual machines belonging to the job. Each VM statically sets its network configuration (IP, hostname, ...) according to the IDs received from the master.

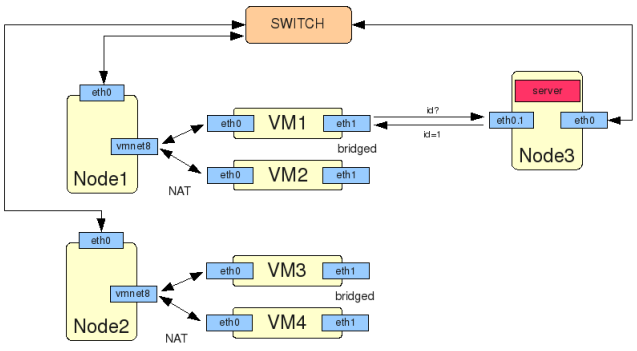


Figure 2: Network configuration

As shown in Figure 2, all the VMs are configured with two NICs: the eth0 NIC exploits the NAT mode provided by the hypervisor and receives it's IP from the local DHCP server of the host machine. The eth1 NIC exploits the bridged mode and it is configured thanks to the ID received from the master on the eth0 NIC. Since one VM network is required

for each reservation, the subnet is based on a unique identifier of each reservation, the ID received from the master node is exploited as the last digit of the IP.

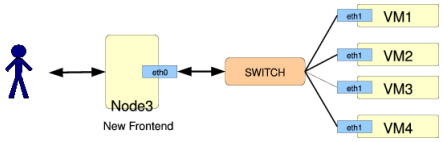


Figure 3: Access to the VMs

The concept of a master node per job becomes more and more important in our proposal. Indeed, in addition to solve the network configuration issue, we exploit it as the dedicated repository to save the snapshots. The master node acts as a new frontend from where we can directly access all the virtual machines and run jobs inside them (see Figure 3).

3.2.2 Taking into account VM migration at grid level

Each VM migration at grid level (ie, between two distinct sites) implies an ARP protocol issue that we have to take into account. All active connections of the migrated VM should be migrated with the VM, ie, if $VM1$ is sending a packet to $VM2$ on the same cluster, the migration of $VM2$ to another cluster should be transparent for $VM1$. The idea of mobility in grids is a well-known issue:

- *mobile IP*, this the first approach to solve the mobility issue. it consists of exploiting a fixed agent. Roughly, when a node moves from a network to another, it informs the agent of the destination network that it is joining the group. The agent establishes a tunnel with the previous agent of the source network. When packets arrives on the originate network, they are redirected by the old agent to the new network. The IPV6 protocol integrates natively the mobility of nodes whereas specific addons are required for IPV4. In both cases, the mobile services should be enabled by the user of the VM and cannot be automatically configured by a generic framework (dependence between OS, security issues, ...)
- *VLAN at grid level*, the VLAN concept is commonly used at cluster level. It enables the creation of virtual networks in the same network. The advantage is that the computers in a Vlan are isolated from the others just like a LAN. The benefits will be complete level 2 isolation enabling for instance to exploit ARP flooding techniques in case of live migration from one site to another one. From our best knowledge, any tool is currently available. The Grid'5000 consortium is actively working on the extension from cluster to grid level of the G5K KaVlan tool.

Network issues due to live migration from one site to another site is not addressed in the current prototype.

3.3 VM Management During Execution

In this section, we analyze the technical issues to tackle in order to transparently monitor, snapshot, and suspend/resume VMs.

3.3.1 VM Monitoring

In order to determine when VMs have to be migrated suspended or resumed, our framework should be able to access information grid resources. Keeping in mind, that designing and developing a complete framework is a tedious and a complex task, we choose to exploit monitoring utilities already deployed in grid infrastructures (such as Ganglia or Nagios system) to collect this information. This choice implies to develop a wrapper for each monitoring system.

3.3.2 Distributed Suspend/Restart

Checkpointing, suspending and restarting parallel programs are well-known complex problems. A typical scenario consists of a job spread over multiple VMs communicating with each other. In that particular case, we have to ensure that suspend and resume operations of the pool of VMs should be transparent from the communication point of view. In other words, all VMs involved in a job should be suspended in a parallel manner, allowing them to be resumed as if they were never interrupted.

Thanks to their external position with regards to VMs, hypervisors can store the entire state (including the network state) of any guest domain running on top of it. Within the assumption that network communications between distributed processes use a reliable protocol and that the suspend/restart of all VMs can be done before fatal timeouts, it is possible to save all the VMs with a coherent network state.

3.3.3 VM scheduling

An advanced management implying a lot of live migrations, suspend/resume operations to optimize particular criterion (such as performance, energy consumption, ...) is beyond the scope of the paper. We emphasize that one of our objective is to address the dynamicity issue in cluster and grid usage in its globality. Each mechanism could be separately address in futur work.

4 Implementation and Experiments

Based on the former analysis, we designed and implemented a first prototype. This framework, entitled `VMdeploy`[1], solves the best-effort issue in G5K. We voluntary choose to implement a non-intrusive prototype: `VMdeploy` interacts with the OAR scheduler to forward best-effort requests and exploits `Kadeploy` when the deployment on an hypervisor is required.

Due to the fact that we "simply" interact with OAR to retrieve computation slots for the best-effort jobs, we do not know when OAR revokes them (and thus kills submitted best-effort jobs). So, there is no way to cleanly suspend the job before to revoke it. As a consequence, our framework makes periodical snapshots of the different VM deployed (see Figure 4) and each snapshot is physically stored on the master node as previously described in Section 3.2.1. Figure 4 presents the architecture of `VMdeploy`. It is composed of two major elements: the best-effort wrapper (FW) and the job manager. The first one provides a dedicated API to submit best-effort jobs and to specify execution constraints (`Kadeploy` environment to convert or VM image to exploit, hardware constraints, ...). The second one is in charge of the VM/job life cycle (currently snapshots and resume operations). The job manager periodically asks the OAR service to get the job status. When it detects that one job has been killed it launches the process of restoring the corresponding VMs. The `VMdeploy` framework submits a new best effort request

[1] https://www.grid5000.fr/mediawiki/index.php/VMdeploy

to the OAR service in order to get a new computation slot where it resumes the latest snapshot taken.

By leveraging `TakTuk`, we provide the possibility to simultaneously suspend several VMs: the job manager opens multiple SSH sessions simultaneously and suspends all the VMs concurrently

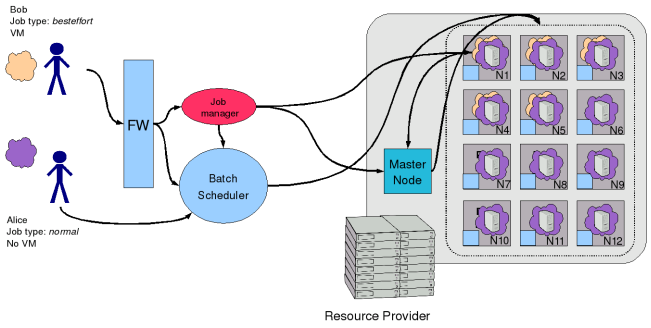


Figure 4: Architecture Overview

4.1 Experiments

In order to evaluate the cost of using `VMdeploy`, we conducted two experiments. The first one analyzes the cost of setting up the whole framework for a job (including the deployment of the hypervisor image to manage VMs on each node). The second one focuses on the snapshot overhead from time and network traffic point of views.

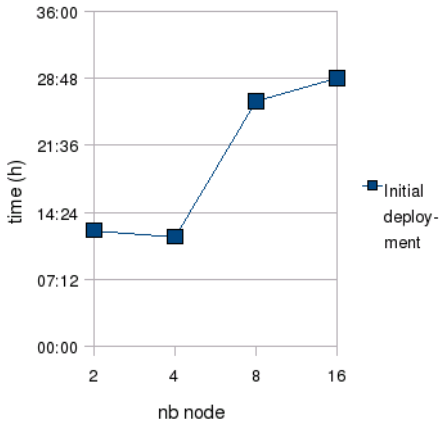
Experiments have been done on the Paravent cluster from Rennes Grid'5000 site. The cluster is composed of HP ProLiant DL145G2 machines, with AMD Opteron 246 2.0GHz CPUs, 2G of RAM memory, two Gigabit Ethernet cards and 80 GB SATA hard drives. The size of the exploited VM image is approximately 400MBytes In our experimentations, 2, 4, 8, and 16 VMs have been used on respectively 2, 4, 8 and 16 nodes.

4.1.1 Initial Deployment

This experiment corresponds to the execution of a best-effort job through `VMdeploy`. First, a default image providing an hypervisor is physically deployed on nodes thanks to `Kadeploy`. Second, the default VM is deployed on each node taking part of the experimentation.Third, the VM configuration process starts (VM boot and network configuration). Finally best-effort job is launched. Figure 4.1.1 gives the different time for each step.

These first results show that the deployment time of the `Kadeploy` hypervisor environment is quite consumming (around 10 minutes). However, we emphasize that this step is going to disappear since the use of hypervisor becomes more and more common (the Grid'5000 default environment will include XEN shortly).

Concerning the VM deployment, the time to copy the VM is significantly growing from 8 images (more than 10 minutes). These results does not correspond to our expectations. First analysis shown an important network traffic from the master node. These results require more investigations.

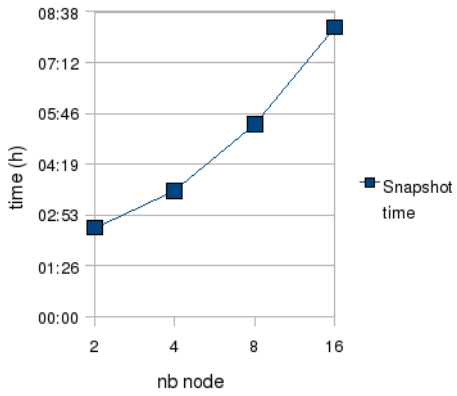


nb node	2	4	8	16
Hypervisor deployment	10:31	09:41	12:36	10:27
VM copy	00:27	00:35	12:05	16:24
VM configuration	01:31	01:32	01:51	01:59
Network traffic	824	1409	2979	5537

Figure 5: Initial Deployment Analysis
Time is given in min:sec and network traffic in MB

4.1.2 Snapshot Management

In this section, we analyzed the overhead of a complete snapshot operation from time and network traffic point of views Results are given in Figure 6. As for the initial deployment, the more VM images we have the more expensive the snapshot operation is. This can be explain due to the bottleneck implied by the several copy from physical nodes to the master one.



nb node	2	4	8	16
Snapshot Time	02:31	03:33	05:27	08:12
Network traffic	416	1274	2972	5423

Figure 6: Snapshot analysis
Time is given in min:sec and network traffic in MB

Preliminary results in both experiments show the importance of the storage management in such a framework aiming at dynamically managing VMs at grid level.

5 Related Work

Historically, batch schedulers are used to manage jobs on clusters. The combination of several jobs with several priorities (for instance, best-effort, interactive, or reservation) on the same resources is a well-known issue [17]. The backfilling model [22] belongs to the most interesting solutions of this issue. To improve performance of backfilling, solutions like checkpointing were set up. These solutions are implemented in (i) user space, i.e., it is at the application level (a relinkage with special libraries are generally required) or (ii) in kernel space (with generally the use of a specific OS) [11, 25].

In this context the use of VMs could give us a lot of interesting functionalities. This is the reason why SGE [10] or Moab were upgraded to take advantage of the VM functionalities. However these two projects are cluster-based and do not take into account grid constraints.

Many significant Grid organizations exploit virtualization technologies. Their focus is based on providing the user a highly configurable environment which meets the needs of their application requests. Such approaches have been developed by the GLOBUS alliance, with its Workspace Service [15], and by many other organizations.

Haizea [23] is a system able to manage the overhead of the management of VMs before the start of the reservation. However in this system fault tolerance is not provided and physical nodes need to be homogeneous. In addition, the implementation of Haizea with a real batch scheduler has not been done yet.

Other works are focused on the energy consumption in grids. As performance increases more and more, the power consumption increases too. Several researches are concentrating on finding a way to save power by shutting down the unused system nodes. The problem that arises is when to decide to shut down some nodes. Several strategies have been proposed (IVS [8], CVS [9] and VOVO [19] [7]). In [12] the Xen virtual machine and the VOVO power saving strategy are combined to save power. The idea is to concentrate the virtual machines on a minimal subset of nodes allowing the other unused nodes to be shut down.

Other projects work on the deployment and management of virtual clusters by working on network virtualization and load balancing between physical clusters. For example, the HIPCAL [20] or VIOLIN [14] projects work on network virtualization. VIOLIN combined with VioCluster [21] allows load balancing between physical clusters using virtualization.

In-VIGO [1] and VMPlants [16] allow to create VMs and configure them automatically.

Others works are based on multiple scheduling levels focusing on the communication between the cluster scheduler and the scheduler of the virtual cluster (Condor [25], Maestro-VC [18]).

More generally, all these works focus on a particular issue and does not address the dynamicity challenge in a transparent and a generic way from the user point of view.

6 Conclusion and Future Work

In this paper we have presented the design of `VMdeploy`, a framework for efficient and transparent management best effort jobs in Grid platforms. We have implemented a first prototype of this framework experimented in the context of the G5K experimental Grid platform. `VMdeploy` relies on the use of VM for executing best effort jobs. As killed operations requested by the unmodified OAR Grid scheduler cannot be detected, `VMdeploy` relies on a job manager periodically saving snapshots of best effort jobs. `VMdeploy` relies on existing monitoring tools to detect killed jobs.

Using our framework, users are thus relieved from the management of checkpoints of their applications and less resources are wasted. In preliminary experiments performed with G5K, we have measured the cost of the VM deployment and snapshot operation implemented in our prototype.

We voluntary chose to address the particular case where applications are fatally taken out from resources. Indeed, the different modes of exploiting clusters or grids (i.e. reservation, interactive or best-effort) imply several issues to consider. For instance, a particular user could book several nodes for a dedicated slot ("reservation") without really requiring all requested resources during the whole amount of time and a 'naive' approach based on an event notification when the application completes will be not sufficient to provide a fine management of the distributed architecture. In this particular case, the assignment policy between the resources delivered by the batch scheduler request and job/user real needs requires to exploit complementary mechanisms (such as, for instance, monitoring probes). Such advanced mechanisms will be addressed in a next work. In contrast to other similar environments, VM deploys is able to manage parallel jobs running in different virtual nodes of virtual clusters deployed in a Grid environment. `VMdeploy` paves the way to scientific clouds.

References

[1] Sumalatha Adabala and et al. From virtualized resources to virtual computing grids: the in-vigo system. *Future Gener. Comput. Syst.*, 21(6):896–909, 2005.

[2] Paul Barham and et al. Xen and the Art of Virtualization. Bolton Landing, New York, USA, October 2003. SOSP'03.

[3] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. Technical report, USENIX Association, 2005.

[4] Havard K. F. Bjerke and et al. Tools and techniques for managing virtual machine images. VHPC '08: Proceedings of the 3rd Workshop on Virtualization in High-Performance Cluster and Grid Computing, 2008.

[5] Raphaël Bolze and et al. Grid'5000: a large scale and highly reconfigurable experimental Grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, November 2006.

[6] Nicolas Capit, Georges Da Costa, Yiannis Georgiou, Guillaume Huard, Cyrille Martin, Grégory Mounié, Pierre Neyron, and Olivier Richard. A batch scheduler with high level components. In *Cluster computing and Grid 2005 (CCGrid05)*, 2005.

[7] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centres. In *Symposium on Operating Systems Principles*, pages 103–116, 2001.

[8] G. Chen, K. Malkowski, M. Kandemir, and P. Raghavan. Reducing Power with Performance Constraints for Parallel Sparse Applications. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, page 231, Washington, DC, USA, 2005. IEEE Computer Society.

[9] E. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In Proceedings of the Workshop on Power-Aware Computing Systems, February 2002.

[10] Niels Fallenbeck, Hans-Joachim Picht, Matthew Smith, and Bernd Freisleben. Xen and the art of cluster scheduling. In *VTDC '06: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, page 4, Washington, DC, USA, 2006. IEEE Computer Society.

[11] Paul H Hargrove and Jason C Duell. Berkeley lab checkpoint/restart (blcr) for linux clusters. *Journal of Physics: Conference Series*, 46:494–499, 2006.

[12] Fabien Hermenier, Nicolas Loriant, and Jean-Marc Menaud. Power Management in Grid Computing with Xen. In *ISPA Workshops*, pages 407–416, 2006.

[13] Alexandru Iosup, Hui Li, Mathieu Jan, Shanny Anoep, Catalin Dumitrescu, Lex Wolters, and Dick H. J. Epema. The Grid Workloads Archive. *Future Gener. Comput. Syst.*, 24(7):672–686, 2008.

[14] X. JIANG and D. XU. Violin: Virtual internetworking on overlay infrastructure. In *Parallel and Distributed Processing and Applications*, pages 937–946. Tech. Rep. CSD TR 03-027, Department of Computer Sciences, Purdue University, July 2003.

[15] K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron. Virtual Workspaces in the Grid. September 2005.

[16] 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 *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 7, Washington, DC, USA, 2004. IEEE Computer Society.

[17] Martin W. Margo, Kenneth Yoshimoto, Patricia Kovatch, and Phil Andrews. Impact of Reservations on Production Job Scheduling. In *Job Scheduling Strategies for Parallel Processing*, pages 116–131. 13th Worshop on Job Scheduling Strategies for Parallel Processing, 2007.

[18] Kiyanclar N., Koenig G.A., and Yurcik W. Maestro-vc: a paravirtualized execution environment for secure on-demand cluster computing. In *Cluster Computing and the Grid Workshops*, page 12, May 2006.

[19] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. Dynamic cluster reconfiguration for power and performance. In L. Benini M. Kandemir and J. Ramanujam, editors, *Compilers and Operating Systems for Low Power*. Kluwer Academic, 2002.

[20] Pascal Primet/Vicat-Blanc, Jean-Patrick Gelas, Olivier Mornard, Dinil Mon Divakaran, Pierre Bozonnet, Mathieu Jan, Vincent Roca, Lionel Giraud, and al. HIPCAL: State of the Art of OS and Network virtualization solutions for Grids. INRIA: Research Report, September 2007.

[21] Paul Ruth, P. McGachey, and Dongyan Xu. Viocluster: Virtualization for dynamic computational domains. In *CLUSTER*, pages 1–10, 2005.

[22] Edi Shmueli and Dror G. Feitelson. Backfilling with Lookahead to Optimize the Performance of Parallel Job Scheduling. In *Job Scheduling Strategies for Parallel Processing*, pages 228–251, 2003.

[23] Borja Sotomayor, Kate Keahey, and Ian Foster. Combining batch execution and leasing using virtual machines. In *HPDC'08: Proceedings of the 17th international symposium on High performance distributed computing*, pages 87–96, New York, NY, USA, 2008. ACM.

[24] Taktuk. Taktuk welcome page. Available at http://taktuk.gforge.inria.fr/.

[25] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the condor experience: Research articles. *Concurr. Comput. : Pract. Exper.*, 17(2-4):323–356, 2005.

[26] Geoffroy Vallée, Thomas Naughton, and Stephen L. Scott. Tutorial: System-level Virtualization and Management using OSCAR. Presented at the 5th Annual OSCAR Symposium (OSCAR 2007), May 15, 2007, Saskatoon, Saskatchewan, Canada. Co-hosted with HPCS 2007.

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 0249-6399