Saline: Improving Best-Effort Job Management in Grids

Jérôme Gallard, Adrien Lebre, Christine Morin

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Saline: Improving Best-Effort Job Management in Grids

Jérôme Gallard — Adrien Lèbre — Christine Morin

N° 7055

September 2009

ISSN 0249-6399 ISRN INRIA/RR--7055--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

Saline: Improving Best-Effort Job Management in Grids

Jérôme Gallard* , Adrien Lèbre† , Christine Morin*

Thème : Calcul distribué et applications à très haute performance.
Équipe-Projet PARIS

Rapport de recherche n° 7055 — September 2009 — 20 pages

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

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

Key-words: Grid Computing, Virtualization, Distributed Systems, 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

Saline: Gérer efficacement des travaux de faible priorité dans les Grilles

Résumé : Les technologies de virtualisation de part la flexibilité qu'elles apportent à la gestion des ressources, sont de plus en plus utilisées dans les grilles de calcul. Cependant, le moyen le plus courant d'exploiter les ressources des grilles reste l'utilisation d'ordonnanceurs à exécution par lots (*batch scheduler*). Afin d'obtenir une gestion des ressources plus efficace de la grille, les ordonnanceurs fournissent généralement un mode "d'exécution au-mieux" (*best-effort*) dans lequel les travaux de faible priorité sont exécutés lorsque les ressources sont inoccupées. Ce mode de fonctionnement ne fournit aucune garantie de service et des travaux de haute priorité peuvent à tout moment venir prendre la place de ceux de plus faible priorité. Ce comportement peut entraîner une grande quantité de perte de calculs ou, impose à l'utilisateur d'instaurer dans son travail des mécanismes de sauvegarde de points de reprise (*chekpointing mechanism*).

Dans ce document, nous présentons `Saline`, un système générique et non intrusif permettant de gérer les travaux de plus faible priorité "au-mieux" et à l'échelle de la grille en utilisant des machines virtuelles (VM). Nous discutons des principaux points clés nécessaires à la conception d'un tel système en se concentrant particulièrement sur les capacités d'arrêt et de redémarrage des machines virtuelles (*VM snapshot*) ainsi que sur la gestion de la configuration du réseau de ces VM. Les premiers résultats d'expériences montrent l'intérêt d'un tel type de système pour assurer l'efficacité de l'exécution des travaux à l'échelle de la grille.

Mots-clés : Grilles de calcul, Virtualisation, Systèmes Distribués, Ordonnanceur, Gestion des ressources.

1 Introduction

Grids (federation of clusters) are used for a wide range of applications providing high-performance computing, large storage capacity, and high throughput communication. Although flexibility usage has been improved (deployment of dedicated environments [3, 28], lease concept [25], ...), the most common way of exploiting such distributed architectures still relies on a reservation scheme where a *static* set of resources is assigned to a job (or a user) during a bounded amount of time. This model of using grids leads to a coarse-grain exploitation of the architecture since resources are simply reassigned to another job/user at the end of the slot without considering the real completion of applications. In the best case, the time-slot is larger and resources are simply under used. In the worst one, running applications are withdrawn from their resources, potentially leading to the loss of all performed computations and requiring to execute once again the same request. If the former case is not really critical from the user point of view, the latter requires to deal with checkpointing issues to ensure the progress of the jobs. This is particularly true for the best-effort mode available in most resource management systems (RMS) where idle resources are assigned to users without any guarantee of service or time allocation. To deal with such issues, particular checkpointing mechanisms have been included in some RMSs [10, 27]. However, their methods are strongly middleware or OS dependant as they require either to link applications with dedicated checkpointing libraries or to exploit a checkpointing capable OS. Such constraints limit the locations where the application can be restarted since grids are generally heterogeneous environments. Consequently, best-effort job series are limited since users have to setup complex systems to periodically save results or to automatically resubmit the lost jobs. Developing such a customized framework for each application is not straightforward and a *naive* resubmission of jobs is usually preferred, leading to a significant waste of computation and power consumption.

In this paper, we propose to develop a generic and non-intrusive framework relying on virtualization snapshotting capabilities to efficiently manage best-effort jobs in coordination with any RMS at grid level. Several works show the interest for virtual machines (VMs) in the context of clusters as they provide flexible, isolated and powerful execution environments. Using VM capabilities such as snapshot, migrate, suspend and resume, it becomes possible to provide a more transparent usage of cluster resources (consolidation [11], preemptive scheduling [26], ...). Designing similar techniques at grid level implies facing new challenges [9]. For instance, VM migration in clusters usually relies on a SAN/NAS file system to share images. This particular way of sharing VM images cannot be exploited in grids and dedicated mechanisms have to be designed. Keeping in mind these constraints, we developed a first prototype entitled `Saline`.

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

The remainder of the paper is organized as follows: Section 2 motivates our work by addressing the impact of coarse-grain management of Grid'5000 best-effort jobs and the benefits of the latest VM capabilities in such a context.

Section 3 is dedicated to the VM management challenges at grid level. Section 4 presents an overview of our prototype, `Saline`, and discusses several experiments focusing on internal mechanisms. Related work is addressed in Section 5. Finally, Section 6 concludes and gives several perspectives.

2 Motivations

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 address these issues.

2.1 Best-effort Jobs in Grid'5000

The *best-effort* concept has been suggested in RMS to back-fill clusters or grids during free slots which do not fit well time-bounded requests. This particular mode does not provide any guarantee of service 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 the potential loss of the performed computations. This behaviour becomes more and more significant as the number of best-effort jobs increases in a large cluster or in a grid. To show the importance of taking into consideration the best-effort issue, we made some statistics on the Grid'5000 traces available from the Grid Workloads Archive [12].

The Grid'5000 architecture (G5K) aims at building a highly reconfigurable, controllable and monitorable experimental Grid platform gathering 9 sites geographically distributed in France featuring a total of 5,000 processors. This is a heterogeneous architecture exploited by multiple users and several kinds of applications. The G5K resource management system relies on the OAR batch scheduler [4] and the `Kadeploy` software [3] which enables to deploy any user environment directly on bare hardware. Unfortunately, it does not provide any feature for suspending or restarting the deployed images.

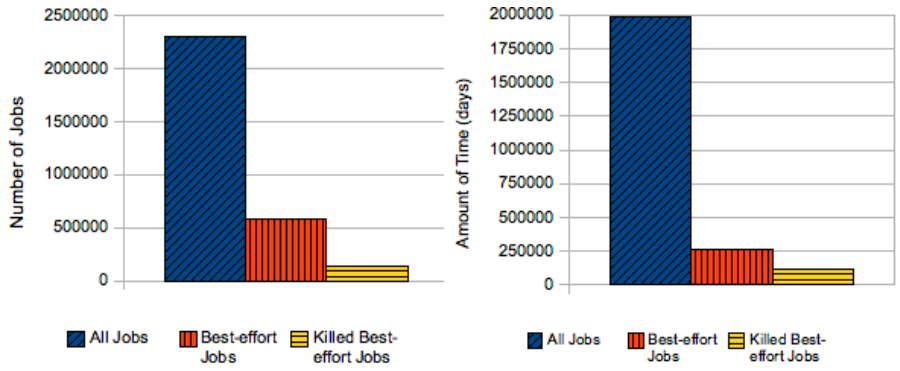
The traces gathered since September 2005 and summarized in Figure 1 show that best-effort jobs are quite used on G5K (25% of the G5K jobs) and 22% of them are simply withdrawn by the OAR scheduler leading to more than 100,000 wasted days of computation (43% of the whole computation time allocated to the best-effort jobs).

As VM technologies offer relocation capabilities, we propose to exploit them to avoid computation loss and correct this particular weakness by running best-effort jobs inside VMs. Thus, it is possible to relocate them from one cluster to another one according to the RMS allocation changes.

2.2 Benefits of Virtualization Technologies

Virtualization is an active research subject since the 70's. However, 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 such as, for instance, Xen [1].

VM technologies provide flexible and powerful execution environments, offering isolation, security and snapshotting mechanisms, customization and encap-



a) Distribution of the Grid'5000 jobs since September 2005
25% of best-effort jobs representing more than 260000 days.

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

b) Best-effort details: 43% 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

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

3 Transparent VM Management at Grid Level

Keeping in mind that we want to design a generic and non-intrusive framework (it should be able to run without requiring any modifications of the target RMS), we established several constraints to be able to submit and manage best-effort jobs until completion through transparent encapsulation into VMs [9]. In our case, a typical scenario of best-effort job consists of an application potentially spread over multiple VMs communicating with each other at cluster level. Thanks to VM capabilities, it is possible to relocate a set of VMs in a coherent network state, from one site to another, assuming we use a reliable network protocol (*e.g.* TCP) and all the VM snapshots are done before fatal timeouts [7]. As mentioned previously, the challenge of managing best-effort jobs is moved to the problem of managing VM through the whole Grid. In that sense, we choose to discuss two major points that we clearly identified as challenges at the grid level: (i) the storage management of VM images during the best-effort job execution and (ii) the network configuration and mobility.

3.1 VM Storage Management

The VM storage management mainly concerns the way in which physical nodes access to the VM images. If disk-less approaches (based on SAN or NAS solutions) can be exploited at cluster level to make VM image management easier, new strategies have to be proposed at grid level. In this paper we consider that VM image is stored on the local hard drive of the node where the VM is currently running. Such a choice implies to consider three major issues: (i) the transfer of the VM image to the right nodes when a new job is launched, (ii) the snapshot of the VM images when the RMS is going to withdraw some resources and (iii) the resume of each VM involved into a job when new resources become available. In addition, in a fourth part, we present a mechanism to improve snapshot management efficiency.

3.1.1 VM Initial Deployment

The VM deployment issue can be compared to previous OS deployment challenge: providing and setting up an OS from one node to several others. In the case of VMs, the issue consists to deploy a VM image from one dedicated repository to several nodes. It can be divided in two distinct phases: (i) the deployment of an appropriate hypervisor on the target nodes and (ii) the deployment of the VM images.

Concerning the hypervisor deployment (i), the framework should provide a generic layer to request hypervisor deployments when required. In this paper, we rely on existing tools like `Kadeploy` [3] to do this operation. Moreover, the current trend in the use of VM technologies in grids lets us think that nodes will already provide a hypervisor, leading to just skip this step.

Concerning the VM image deployment (ii), we propose to use existing tools to make *efficient VM image copy from a dedicated repository to the target nodes.* For instance, the `Taktuk`[1] utility [6] is able to dynamically deploy the VM image from a central repository to the target nodes.

3.1.2 Saving of VM Snapshots

Various strategies could be implemented for saving the VM snapshots, *e.g.*, saving them locally or on a remote node [9]. Each method having its own strength and weakness, in this paper, we focus on the remote method *i.e.* saving all the VM snapshots on a dedicated repository.

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

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

[1] http://taktuk.gforge.inria.fr/

job should be migrated on the new resources. Otherwise, all the VMs should be suspended to the VM repository (ii).

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

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

- *copy one VM image from the node containing the lowest number of VM images*[2] *to the repository.*
- *from each other node, copy VM images to another node except if:*
  - *the destination node is already sending or receiving a VM image,*
  - *the destination node has less VM images than the sender node,*
  - *the sender node is already sending or receiving a VM image.*

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

3.1.3 Deployement of VM Snapshots

The resume operation of each job requires to deploy the latest snapshots on the new allocated nodes. This issue cannot be simply compared to the intial deployment problem. Indeed, in that particular case, distinct images have to be transferred from the repository to the n nodes involved into the job and potentially belonging to a remote cluster.

3.1.4 Snapshot Improvement: Copy-On-Write

Copy-On-Write techniques [2] consist in saving only each VM image modification instead of each whole VM image. Indeed, with Copy-On-Write each VM is represented by a reference VM image and a *diff* file. The *diff* file contains all the changes made by the VM (*i.e.* file system) toward the reference VM image. In addition, the *diff* file contains the volatile state of the VM (the VM's memory). Indeed, to save the volatile state of the VM consists in serializing its memory in the *diff* file. In this manner, in our case, only the *diff* file of a VM should be saved when a snapshot is requested.

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

3.2 Network Configuration and Mobility

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

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

MAC address configuration: MAC addresses are composed of 48 bits (see Table 1). Each NIC has a MAC address in which the first 24 bits (MAC1) are used for the manufacturer ID and the last 24 bits (MAC2) are unique to the NIC. In our case, we define our own manufacturer ID for MAC1. MAC2 ($\approx$ 16 millions of distinct addresses) is used to distinguish each VM NIC. We compose the first 16 bits of MAC2 with the *subnetID* received from the master. The last 8 bits of MAC2 are set according to the *vmID*.

<table>
<tr><td colspan="3">48 bits</td></tr>
<tr><td rowspan="2">24 bits
MAC1: manufacturer ID</td><td colspan="2">24 bits
MAC2: unique to the NIC</td></tr>
<tr><td>16 bits
subnetID</td><td>8 bits
vmID</td></tr>
</table>

Table 1: MAC address bitmap composition.

IP address configuration: Each VM statically sets its network configuration according to the two IDs received from the master (see Table 2). We assume the target grid platform allows the use of the 10.0.0.0/8 network. IP addresses are composed of 32 bits. In the 10.0.0.0/8 network, the first 8 bits (IP1) are assigned by the IANA Authority[3]. The second 24 bits are used by our framework. In these 24 bits, the first 16 bits (IP2) are set to the *subnetID* and the last 8 bits (IP3) are set to the *vmID*. This allow our framework to choose between 65536 networks (from 10.0.0 to 10.254.254) and 254 VMs (from 1 to 255). Of course, if the number of VMs is greater than 254, it should be possible to aggregate several subnets.

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

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

Table 2: IP address bitmap composition.

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

4 Implementation and Experiments

Based on the previous analysis, we implemented a prototype, entitled `Saline`[4], which aims at solving the best-effort issue in G5K. In the current version, it interacts with the OAR grid scheduler.

4.1 Implementation

Figure 2 presents the architecture of `Saline`. From the grid point of view, `Saline` is composed of several instances (one per site). Each `Saline` instance relies on two major elements: the best-effort wrapper (BEW) and the job manager. The first one provides a dedicated API to submit best-effort jobs and to specify execution constraints (VM image to exploit, hardware constraints, ...). The second one is in charge of VM management, periodically: (i) making snapshots of the current running VMs and saving them on the master node (i.e, the image repository), and (ii) asking the OAR service for the job status. When it detects that one job has been killed it launches the process of restoring the corresponding VMs: it submits a new best-effort request to OAR. If OAR is not able to find available resources on the whole grid, `Saline` waits a few minutes before asking OAR again. If OAR finds new resources, theses resources could be in the original site (cluster level) or, in another site (grid level). In the cluster level case, `Saline` has to just redeploy snapshot images from the master to the nodes belonging to the same site. In the grid level case, `Saline` has to redeploy snapshot images on nodes of another site. This operation requires two steps: first, copying the considered snapshot images on the distant nodes, second, updating the distant `Saline` master to take into account the arrival of the best-effort jobs from another site.

Concerning the network, the configuration of the NIC IDs for one VM is described in Figure 3. Obviously, this method can be used for multiple VMs running on top of one or more physical nodes. All the VMs are configured with

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

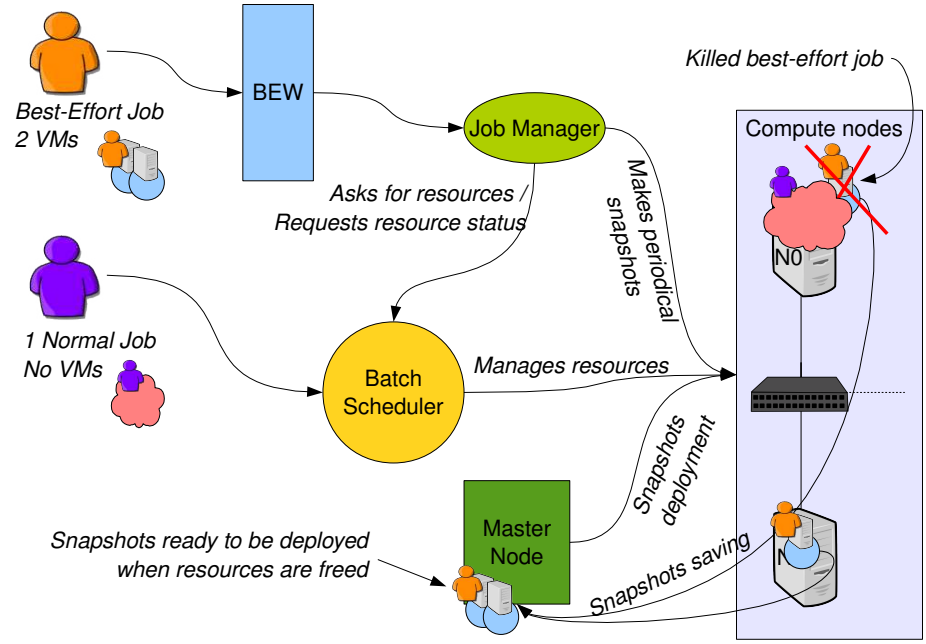
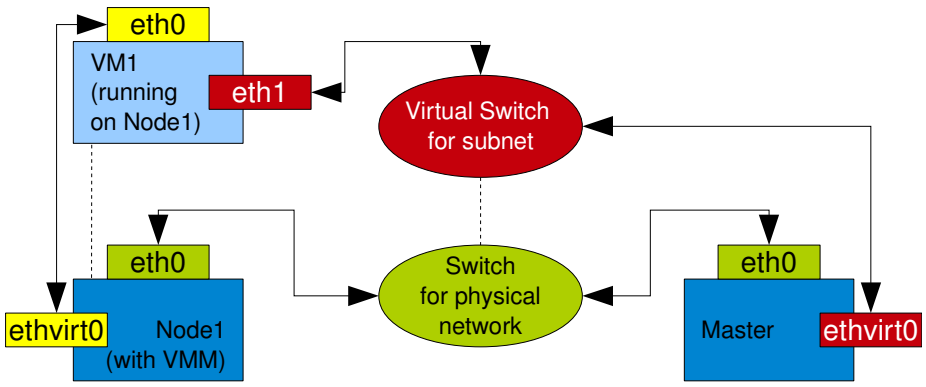


Figure 2: `Saline` Architecture Overview

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



VM1 is a VM running on the top of *Node1*. 1) *VM1* *eth0* NIC is configured in NAT mode thanks to the hypervisor of *Node1*. 2) *VM1* asks to the *Master* a *subnetID* and a *vmID* via its *eth0* NIC. 3) *VM1* configures its *eth1* NIC in bridge mode with the IDs provided by the *Master*. This creates a subnet. 4) From the *Master*, it is possible to communicate directly with all VMs of the jobs (via its *ethvirt0* NIC) and all VMs could communicate directly with others (via their *eth1* NIC).

Figure 3: VM Network Configuration Principle

4.2 Experiments

In order to evaluate the cost of using `Saline`, we conducted four sets of experiment. The first one analyzes the cost of setting up the whole framework for a job. The second one focuses on the overhead of the saving snapshot both with and without notification. The third one studies the deployment snapshot overhead. The last one deals with the inter-site migration in case of notification callback usage. As we want to focus on grid aspect, we chose to not discuss intra-site migration where best-effort jobs can be migrate to other nodes in the same cluster.

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

4.2.1 Initial Deployment

The initial deployment could be decomposed in several parts. First, a default image providing a hypervisor is physically deployed on nodes using `Kadeploy` (i). Second, the VM image is deployed on each node taking part in the experiment (ii). Third, the VM configuration process starts (iii). Finally the best-effort job is launched.

Concerning the hypervisor deployment (i), results show that the deployment of the `Kadeploy` hypervisor environment is quite time consuming ($\approx$ 10 minutes for $\leq$ 64 nodes). However, we emphasize this step is going to disappear since hypervisors will already be setup on the nodes.

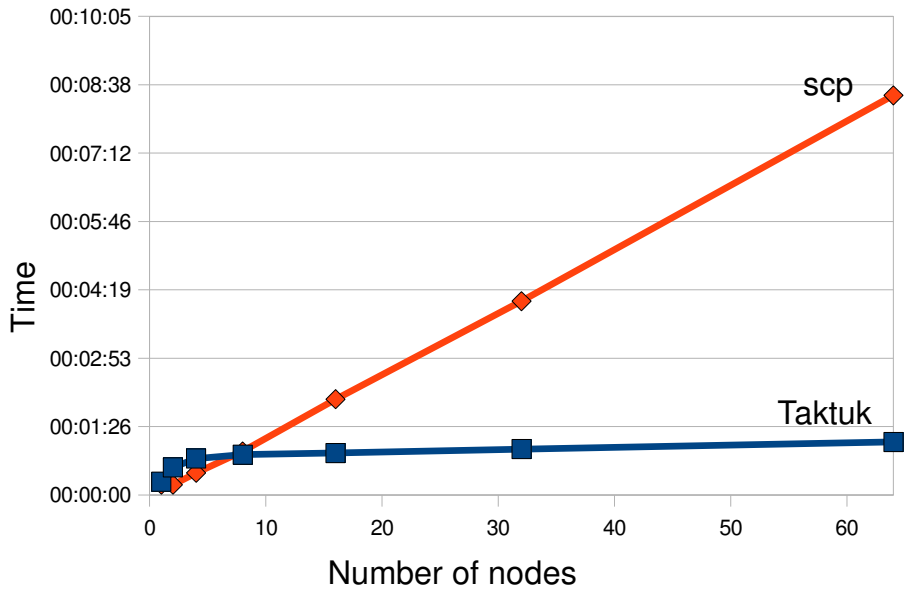


Figure 4: Initial Deployment Analysis

Concerning the VM image deployment (ii), in Figure 4, we compare the use of multiple simultaneous *scp* to an efficient copy mechanism `Taktuk`. The X-axis corresponds to the number of nodes used in the experimentation and the Y-axis corresponds to the time required to copy all the VM images. Results show, that with `TakTuk`, the time to copy the VM image from the master to all compute nodes, is not directly linked to the number of nodes. Indeed, using `Taktuk`, the VM image copy takes less than one minute (whatever the number of nodes) whereas for multiple `scp` copies, the number of nodes is directly linked with the time to complete all the copies (for 64 nodes, it takes more than 8 minutes). This very good performance is due to the fact that `TakTuk` is able to pipeline the copy of the VM images from the master to the nodes.

Concerning the VM configuration (VM boot and network configuration) (iii), results show this step could be neglected compared to the others. Indeed, it takes less than 2 minutes ($\leq$ 64 nodes) to configure all the VMs.

4.2.2 Snapshot Saving

With or without notification, the problem of snapshot management mainly concerns the copy of n VM images to one master which implies a network bottleneck at the master side. Indeed, without notification, the framework has to save periodically the VM snapshots, and with notification, the framework has to save the VM snapshots at a particular time. This section deals with (i) the overhead of the saving snapshot (both with and without notification callbacks) and (ii) the optimization proposed in Section 3.1.2 to efficiently retrieve the snapshot images.

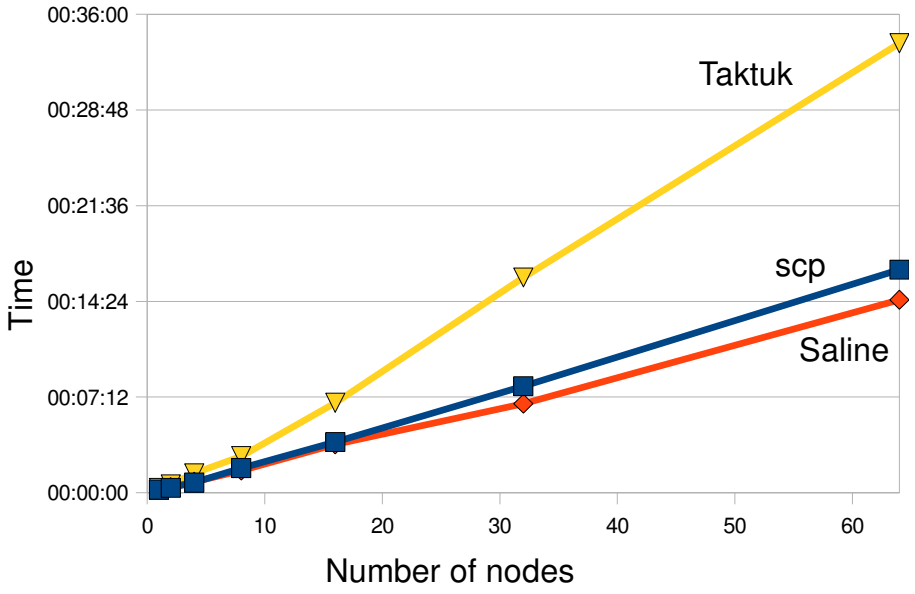


Figure 5: Snapshot Saving Analysis

Snapshot saving overhead (i). We compared three solutions: one copying with `scp`, another using `Taktuk`, and the last with our proposal implemented in `Saline`. Results visible in Figure 5 show that `Saline` has similar ($\leq$ 16 nodes) or better result than the two other tools to copy VM snapshots from several nodes to the master. Indeed, `Saline` is able to manage the network bandwidth of the master by limiting the number of packet collisions and lost packets.

Optimization in case of notification (ii). In addition, Figure 6 presents the percentage of node freed according to the time. The X-axis presents the time and the Y-axis presents the percentage of nodes freed. For this experiment, we used 64 nodes and measured how long it took to free each node. Results show clearly that, `Saline` is able to free 80% of all nodes in less than 2 minutes whereas other tools do the same after 17 minutes for `scp` and 29 minutes for `Taktuk`.

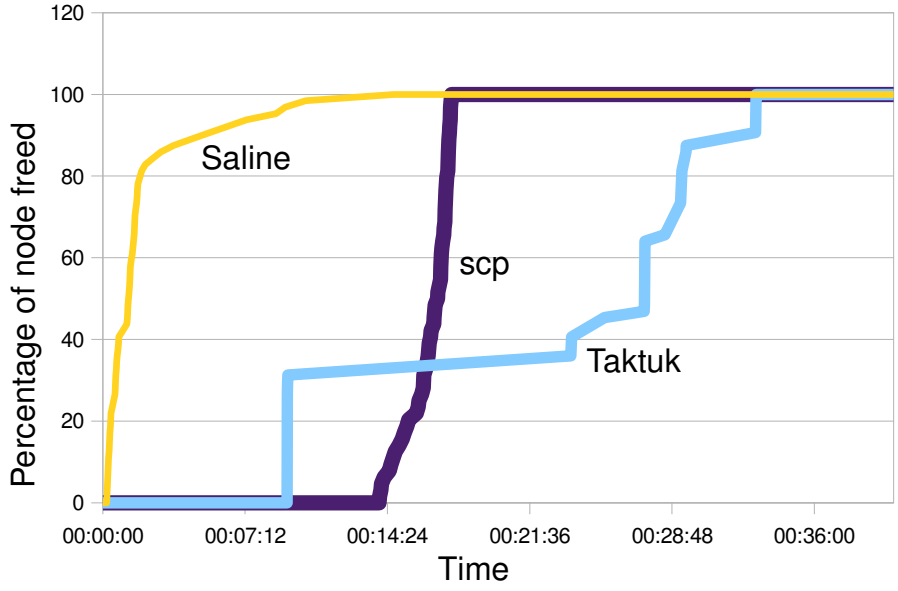


Figure 6: Detailed Snapshot Time Analysis

4.2.3 Snapshot Deployment

In this experiment we first deployed snapshot images at cluster level (Sophia site) and secondly at grid level (from Sophia to Nancy site) Figure 7 presents the required time to deploy all VM snapshots in both cases. As mentioned earlier, such a development cannot be efficiently solved by using `Taktuk`. We currently exploit simultaneous `scp` commands to transfer VM snapshots from the repository node to the destination ones. Results show that the more physical nodes we have, the bigger the time to deploy all snapshots is. However, in the case of best-effort jobs, the time to deploy all VM snapshots ($\approx$ minutes) compared to the time to restart the best-effort job since the beginning ($\approx$ hours or days) is negligible. Moreover, results show that the time to restart snapshot images

at cluster level is nearly the same as the time at grid level. This result could be explained by the network topology of G5K. Indeed, the network bandwidth between Sophia and Nancy is 10GB, whereas on a site, the network bandwidth between one node and its router is 1GB. In that condition, the number of nodes we use is negligible compared to the network bandwidth.

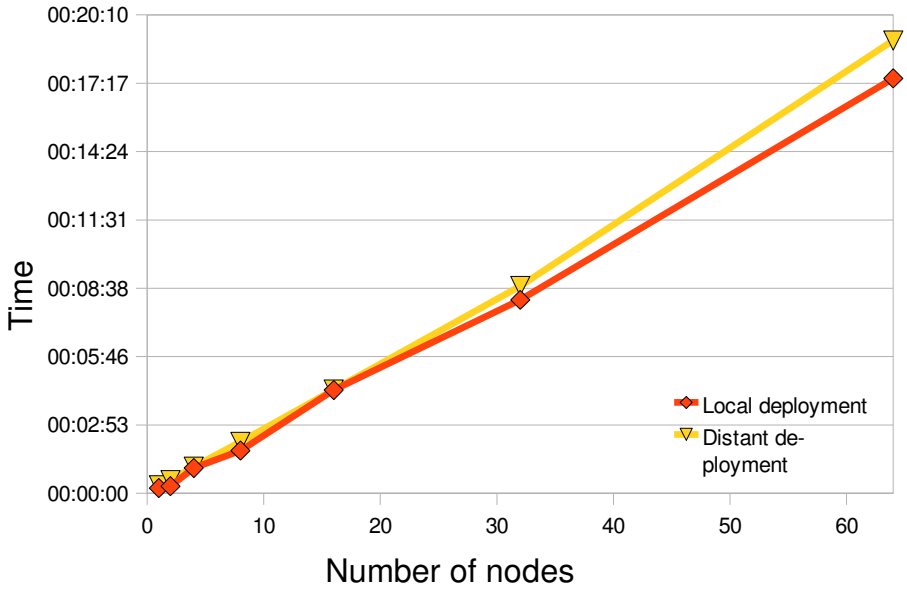


Figure 7: Snapshot Deployment

4.2.4 Inter-Site Migration

In this section, we study the cost of VM migration from n node of one site to n node of another one in case of notification. Experiments were done between the sites of Sophia and Nancy.

Table 3 shows the result of the experiment. Clearly the time to migrate VMs from one site to another is negligible ($\approx$ seconds) compared to the time to snapshot VM images ($\approx$ minutes) or to restart them from the beginning ($\approx$ hours or days). As already said in the previous Section, this very interesting result could be explained by the network topology of G5K.

nb nodes	1	2	4	8	16	32	64
Time	00:18	00:20	00:21	00:22	00:23	00:25	00:32

This table shows the time required to copy n VM snapshots from n nodes to n nodes of another site (time is given in min:sec).

Table 3: Inter-Site VM Migration

5 Related Work

Historically, batch schedulers have been used to manage jobs on clusters. The combination of several jobs with several priorities on a limited number of resources is a well-known issue [17]. The back-filling model [24] is one of the most interesting solutions to this issue. To improve performance of back-filling, solutions like checkpointing were set up. These solutions are implemented in (i) user space, i.e., at application level (a linkage with special libraries is generally required) [20, 22] or (ii) in kernel space (generally using specific OS) [10, 27].

In such a context, the use of VM snapshot capabilities could make application management easier. This is the reason why SGE [8] or Moab were upgraded to take advantage of VM functionalities. However these two projects are cluster-based and do not take into account grid constraints.

Many significant organizations focus on providing a highly configurable environment, which meets the needs of the user application requests. Such approaches have been developed by the GLOBUS alliance with its Workspace Service [14]. However, they do not provide an easy and transparent way to manage resources dynamically.

Concerning the Cloud area, some open-source projects like Nimbus [15], Eucalyptus [18] or SnowFlock [16] allow users to deploy their cloud and manage a virtual cluster on a physical one. However, to our knowledge, such systems don't deal with VM snapshot and/or migration at grid level.

Other works are focused on using VM capabilities to reduce energy consumption. As performance increases more and more, power consumption increases too. Several researches are concentrating on finding a way to save power by shutting down unused system nodes (IVS [5], VOVO [19] and Entropy [11]). However, to our best-knowledge all these solutions do not consider grid and are only able to do migration of jobs at cluster level.

Other projects focus on the deployment and management of virtual clusters by working on network virtualization (HIPCAL [21] or VIOLIN combined with VioCluster [13, 23]). However and once again, these works focus on a particular issue and do not address the VM issue at grid level.

6 Conclusion and Future Work

In this paper, we address the challenge of managing best-effort jobs by encapsulating them into VMs and thus relieve users of dealing with traditional checkpointing issues. In addition and thanks to VM capabilities such as the snapshotting and the remote resume ones, it is possible to manage resources in a more efficient way through transparent relocations between clusters of a grid.

We discussed two major issues that should be addressed at grid level: the management of VM images during the whole execution of the best-effort job and the network configuration and mobility of each VM involved into a job.

After discussing these points, we present our framework `Saline`, a generic and non-intrusive framework to deal with best-effort jobs in coordination with any RMS. Thanks to it, best-effort jobs can be launched into VMs and be transparently relocated from one cluster to another one each time it is required. `Saline` is able to benefit from RMS notification callbacks. In this case, `Saline` tries to migrate the VMs involved into the job. If no resources are available on

another site, it makes the snapshot of the whole set of VMs, and waits until new resources become available. If no callbacks are present, `Saline` simply makes periodical snapshots of the set of VMs and checks their status. If VMs are not reachable *i.e.* resources have been withdrawn, `Saline` requests the RMS for new ones and restarts the job from the most recent snapshot.

First experiments on our prototype show promising results with regard to current lost of computation in a grid such as Grid'5000.

Now, we are extending our framework by exploiting complementary mechanisms like monitoring probes. Indeed, which such probes, `Saline` will be able to optimize its scheduling decision (load-balancing between nodes/sites, prediction of hardware failure, ...). In the meantime, we are currently evaluating new mechanisms to improve performances during the snapshot redeployment.

Concerning the management of the VMs, some points need to be addressed: (i) the possibility to have a job spread on multiple sites at the same time, (ii) the possibility to migrate a subset of VMs of a job from one site to another on a transparent way for the whole set of VMs and (iii) the possibility for VMs to communicate with clients outside the grid. To solve these issues we envision using network virtualization. More generally, we addressed the best-effort job issue in that work. However, we think that it could be easily extended to all kind of jobs and thus, could contribute to make the use of scientific clouds easier.

Contents

References

[1] 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 9th ACM symposium on Operating systems principles (SOSP)*, New York, NY, USA, 2003.

[2] Havard K. F. Bjerke, Dimitar Shiyachki, Andreas Unterkircher, and Irfan Habib. Tools and techniques for managing virtual machine images. In *Proceedings of the 3rd Workshop on Virtualization in High-Performance Cluster and Grid Computing (VHPC)*, 2008.

[3] Raphaël Bolze and et al. Grid'5000: A Large Scale and Highly Reconfigurable Experimental Grid Testbed. *International Journal of High Performance Computing Applications*, 2006.

[4] 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 *Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, 2005.

[5] G. Chen, K. Malkowski, M. Kandemir, and P. Raghavan. Reducing Power with Performance Constraints for Parallel Sparse Applications. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Washington, DC, USA, 2005.

[6] Benoit Claudel, Guillaume Huard, and Olivier Richard. TakTuk, adaptive deployment of remote executions. In *Proceedings of the 18th ACM international symposium on High performance distributed computing (HPDC)*, pages 91–100, New York, NY, USA, 2009.

[7] Wesley Emeneker and Dan Stanzione. Increasing Reliability through Dynamic Virtual Clustering. In *High Availability and Performance Computing Workshop*, 2006.

[8] Niels Fallenbeck, Hans-Joachim Picht, Matthew Smith, and Bernd Freisleben. Xen and the art of cluster scheduling. In *Proceedings of the 2nd Int. Workshop on Virtualization Technology in Distributed Computing (VTDC)*, 2006.

[9] Jérôme Gallard, Oana Goga, Adrien Lèbre, and Christine Morin. VMdeploy: Improving Best-Effort Job Management in Grid5000. Technical Report RR-6764, INRIA, December 2008.

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

[11] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy: a Consolidation Manager for Clusters. In *Proceedings of the International Conference on Virtual Execution Environments (VEE)*, 2009.

[12] 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.

[13] 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, 2003.

[14] Kate Keahey, Ian Foster, Tim Freeman, Xuehai Zhang, and Daniel Galron. Virtual Workspaces in the Grid. In *11th International Euro-Par Conference, Lisbon, Portugal*, 2005.

[15] Kate Keahey and Tim Freeman. Contextualization: Providing One-Click Virtual Clusters. In *Inter. Conference on e-Science*, 2008.

[16] H. Andres Lagar-Cavilla, Joseph Whitney, Adin Scannell, Philip Patchin, Stephen M. Rumble, Eyal de Lara, Michael Brudno, and M. Satyanarayanan. SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing. In *the 3rd European Conf. on Computer Systems (Eurosys)*, Germany, 2009.

[17] Martin W. Margo, Kenneth Yoshimoto, Patricia Kovatch, and Phil Andrews. Impact of Reservations on Production Job Scheduling. Workshop on Job Scheduling Strategies for Parallel Processing, 2007.

[18] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus Open-source Cloud-computing System. In *the 9th Inter. Symposium on Cluster Computing and the Grid (CCGRID)*, Shanghai, China, 2009.

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

[20] James S. Plank, Micah Beck, Gerry Kingsley, and Kai Li. Libckpt: Transparent Checkpointing Under Unix. In *Proceedings of the USENIX 1995 Technical Conference (TCON)*, pages 213–223, Berkeley, CA, USA, 1995.

[21] 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.

[22] Joseph F. Ruscio, Michael A. Heffner, and Srinidhi Varadarajan. DejaVu: Transparent User-Level Checkpointing, Migration and Recovery for Distributed Systems. In *Supercomputing (SC)*, New York, NY, USA, 2006.

[23] Paul Ruth, P. McGachey, and Dongyan Xu. VioCluster: Virtualization for Dynamic Computational Domains. In *CLUSTER*, 2005.

[24] 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.

[25] 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 (HPDC)*, New York, NY, USA, 2008. ACM.

[26] Mark Stillwell, David Schanzenbach, Frédéric Vivien, and Henri Casanova. Resource Allocation Using Virtual Clusters. *Cluster Computing and the Grid (CCGRID)*, 2009.

[27] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed Computing in Practice: the Condor Experience. *Concurr. Comput. : Pract. Exper.*, 2005.

[28] 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