# Cluster-Wide Context Switch of Virtualized Jobs

Fabien Hermenier, Adrien Lebre, Jean-Marc Menaud

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

# *Cluster-Wide Context Switch of Virtualized Jobs*

Fabien Hermenier, Adrien Lèbre, Jean-Marc Menaud

*ASCOLA Research Group*
*EMN, INRIA, LINA UMR 6241*
firstname.lastname@emn.fr

**N° 6929**

Avril 2009

Thème COM


ISSN 0249-6399 ISRN INRIA/RR--6929--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

# Cluster-Wide Context Switch of Virtualized Jobs

Fabien Hermenier, Adrien Lèbre, Jean-Marc Menaud

*ASCOLA Research Group*
*EMN, INRIA, LINA UMR 6241*
firstname.lastname@emn.fr

Thème COM — Systèmes communicants
Équipe-Projet ASCOLA

Rapport de recherche n° 6929 — Avril 2009 — 21 pages

**Abstract:** Clusters are massively used through Resource Management Systems with a static allocation of resources for a bounded amount of time. Such an approach leads to a coarse-grain exploitation of the architecture and an increase of the job completion times since most of the scheduling policies rely on users estimates and do no consider the real needs of applications in terms of both resources and times. Encapsulating jobs into VMs enables to implement finer scheduling policies through cluster-wide context switches: a permutation between VMs present in the cluster. It results a more flexible use of cluster resources and relieve end-users of the burden of dealing with time estimates.

Leveraging the Entropy framework, this paper introduces a new infrastructure enabling cluster-wide context switches of virtualized jobs to improve resource management. As an example, we propose a scheduling policy to execute a maximum number of jobs simultaneously, and uses VM operations such as migrations, suspends and resumes to resolve underused and overloaded situations. We show through experiments that such an approach improves resource usage and reduces the overall duration of jobs. Moreover, as the cost of each action and the dependencies between them is considered, Entropy reduces, the duration of each cluster-wide context switch by performing a minimum number of actions, in the most efficient way.

**Key-words:** Context Switch, Virtualization, Cluster

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

# Changement de conexte pour tâches virtualisées à l'échelle des grappes

**Résumé :** De nos jours, la gestion des ressources d'une grappe est effectuée en allouant des tranches de temps aux applications, spécifiées par les utilisateurs et de manière statique. Pour un utilisateur, soit les ressources demandées sont sur-estimées, et la grappe est sous-utilisée, soit sous-dimensionnées, et ses calculs sont dans la plupart des cas perdus. L'apparition de la virtualisation a apporté une certaine flexibilité quant à la gestion des applications et des ressources des grappes. Cependant, pour optimiser l'utilisation de ces ressources, et libérer les utilisateurs d'estimations hasardeuses, il devient nécessaire d'allouer dynamiquement les ressources en fonction des besoins réels des applications. Être capable de démarrer dynamiquement une application lorsqu'une ressource se libère ou la suspendre lorsque la ressource doit être ré-attribuée. En d'autres termes, être capable de développer un système comparable au changement de contexte sur les ordinateurs standards pour les applications s'exécutant sur une grappe. En s'appuyant sur la virtualisation, développer un tel mécanisme de manière générique devient envisageable.
Dans cet article nous proposons une infrastructure offrant la notion de changement de contexte d'applications virtualisées appliquée aux grappes. Cette solution a permit de développer exécutant simultanément un maximum d'applications virtualisées. Nous montrons qu'une telle solution augmente le taux d'occupation de notre grappe et réduit le temps de traitement des applications.

**Mots-clés :** Changement de contexte, Virtualisation, Grappe

# 1 Introduction

Clusters are used for a wide range of applications providing high-performance computing, large storage capacity, and high throughput communication. According to their size and their objectives, clusters are exploited in different ways. However, few of them are dedicated to one particular application and the most common way of exploiting large cluster consists in using dedicated services especially Resource Management System (RMS) where users request resources for a specific walltime according to their estimated needs.

Several works have been proposed to provide more flexibility to administrators and users ([1], [2], [3],[4], ...). However, cluster usage is still based on a reservation scheme where a static set of resources, including potentially several nodes, is assigned to a job during a bounded amount of time. If one may argue that providing dedicated time slot per job is required in some situations (e.g. in the case of reproducible experiments), a lot of end-users do not really take care about such critical requirements and just want to benefit from clusters as soon as possible and as long as required. As a consequence, this model of using large clusters by only considering resource and time estimates leads to a coarse-grain exploitation of the architecture since resources are reassigned to another job at the end of the slot without considering the real needs of applications in terms of both resources and times. In the best case, the time-slot is larger than the estimate and resources are simply under used. In the worst case, running applications can be withdrawn from their resources leading potentially to the loss of all the performed calculations and requiring to execute once again the same request.

In this paper, we propose to address the issue of static allocations by extending the Entropy consolidation framework [5] with a cluster-wide context-switch mechanism. A well-known approach to improve the resource utilization in clusters consists in exploiting preemption mechanisms where jobs can be proceed, even partially, and suspended according to the scheduler objectives. The RMS performs transitions between the current situation and the expected one: jobs to stop are stopped or suspended to a disk while jobs to run are started or resumed from previously saved images. Such transitions can be considered as a cluster-wide context switch. With virtualized jobs, *vjobs i.e.* jobs encapsulated into Virtual Machines (VMs), it is possible to implement finer job scheduling policies to significantly reduce the loss of computation time and to provide a more transparent cluster usage for end-users: live migration [6] is useful to adapt the assignments of VMs according to their current requirements [7, 8, 9], while the suspend/resume operations provide persistent snapshotting capabilities to consider distinct priorities between jobs [10].

Some works [11, 4] have stated the lack of flexibility in current resource allocation policies but without considering cluster-wide context switch as a fundamental building block to improve cluster resource usage. Providing a such environment require to consider several issues, such as maintaining the consistency of environments during a suspend or a resume of several inter-connected VMs or dealing with dependencies issues between the different actions to perform. In addition, a cluster-wide context switch can be time consuming: migrate a VM takes up to 26 seconds, while resume a VM to a distant node takes up to 3 minutes in our experiments. Thus, similarly to the context switch of a process in a computer, the cost of a cluster-wide context switch has to be evaluated and reduced as possible to minimize its impact on the cluster.

The prototype described in this paper provides the different mechanisms to efficiently manage a cluster-wide context switch of virtualized jobs. Such a framework facilitates the implementation and the evaluation of advanced cluster scheduling strate-

gies based on a VM granularity. As an example, we present a strategy which performs a maximum number of *vjobs* simultaneously according to their running priorities and their resource requirements. By accessing external monitoring information service, Entropy can react according to the cluster load. When some nodes are overloaded, the framework performs VM migrations or suspend the lowest ones to give a sufficient amount of CPU and memory resources to the *vjobs* with a higher priority. Similarly, when the cluster is considered as underused, *vjobs* that were previously suspended are resumed and/or new *vjobs* are started. In order to deal with dependency issues, a dedicated module of Entropy reduce the cost of the whole reconfiguration. In our experiments, a simulation using workload traces shows that our approach can fit with clusters composed of up to 200 nodes and almost 500 VMs. In addition, we find that an execution with a static allocation of 9 *vjobs* involving 72 VMs, implies a total execution time of 250 minutes on 11 nodes while our approach, based on our sample scheduling algorithm using cluster-wide context switches, reduces this time to 150 minutes with a average duration for the context switches equals to 70 seconds.

The remainder of the paper is organized as follows: Section 2 gives a brief overview of the batch scheduler limitations and addresses the cluster-wide context switch of VMs in a general way. Section 3 presents an overview of the architecture. Section 4 describes how the context switch is prepared and how it is optimized to reduce its duration. Experiments are presented and discussed in Section 5. Section 6 addresses related work, and Section 7 concludes this paper and gives some perspectives.

# 2 Principles

## 2.1 Batch Scheduler Limitations

As mentioned earlier, most of the clusters rely on a reservation scheme where traditional batch scheduler assigned a static set of resources to a job during a bounded amount of time. The usual behavior consists in scheduling submitted jobs in a *First Come First Serve* strategy with the EASY backfilling policy. Figure 1 depicts the process: jobs arrive one after the other and are scheduled according to the estimated execution-time and the estimated resource requirements. The backfilling mechanism deals with fragmentation issues while guaranteeing a time reservation for the first job in the queue (Figure 1 (b)). Even if there are more advanced strategies such as the *Conservative* one (maintaining time guarantees for each waiting job in the queue), backfilling approaches have some strong limitations that prevent an optimal usage of the cluster resources without exploiting advanced capacities such as preemption mechanisms. Such a functionality enables to run a job, even partially, each time it is possible (Figure 1 (c)).

Moreover, the users estimates delivered at job submission time are often not accurate from time as from resources point of views [12]. Even if batch schedulers try to backfill as soon as one job has been stopped, backfilling strategies cannot manage application requirement changes. A job may complete before the end of its time slot without referring the RMS that resources can be freed. In addition, "reservation mode" available in the majority of RMS allows users to book more cluster nodes than required without checking if all resources are really exploited during the application runtime. In all of these situations, dynamic scheduling is mandatory to finely exploit cluster resources.

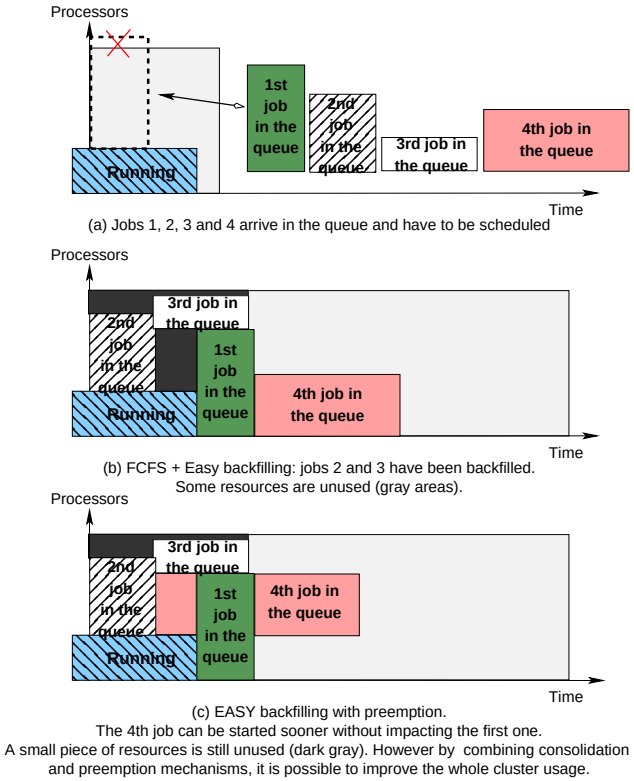



Figure 1: Backfilling limitations

Usually used for fault-tolerance issues, checkpointing solutions, like checkpointing-based resource preemption, have been suggested to provide finer scheduling strategies [13, 14]. However, these methods are strongly middleware or OS dependent. Moreover they do not consider application resource changes. Single System Images such as openMosix or Kerrighed [15] have integrated advanced strategies based on preemptive and migration approaches. Unfortunately, due to the development complexity, most of SSI implementations have not been finalized. Thanks to the latest improvements, virtualization tools could resolve this lack of dynamicity in cluster scheduling policies by specifically using migrate, suspend and resume actions. Each of these operations changes the state of the ”virtualized” job.

## 2.2 Life Cycle of Virtualized Jobs

Relying on the VM abstraction, we propose to reconsider the batch scheduler granularity from the usual job to the virtualized one. A *virtualized job* (a *vjob*) can be spread on one or several VMs.

At the submission, a *vjob* is in *Waiting* state, it will evolve through different states described in Figure 2.

When enough resources are available for each VM that belongs to a waiting *vjob*, the scheduler can execute the action *run* on all the associated VMs and switches the vjob to the *Running* state. In some situations, such as an overloaded cluster, the scheduler can select some running *vjobs* and executes several *suspend* operations. These actions, first, write the memory and the state of each concerning VM on a persistent devices to free resources and second, switches the selected vjobs to the *Sleeping* state.

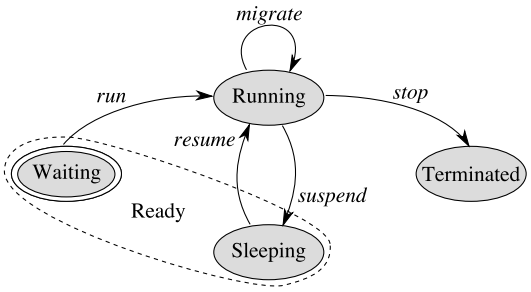
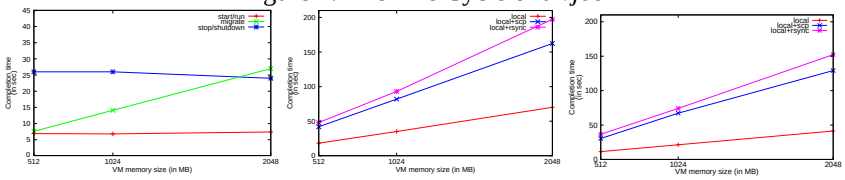



Figure 2: The Life Cycle of a *vjob*




(a) Run/Migrate/Stop (b) Suspend (c) Resume

Figure 3: Duration of each transition (i.e of each VM context switch) according to the amount of memory allocated to the VM

Thus, the pseudo-state *Ready* combines the runnable *vjobs*, that is *vjobs* that are in the *Sleeping* or *Waiting* states. When a *vjob* is considered as finished by its owner, its VMs are removed from the system a *stop* action and the vjob switches to the *Terminated* state. Finally, the *migrate* action does not affect the state of a *vjob*. This operation exploits live migration mechanism to switch a VM from its current host to a new one. This latest transition enables to finely handle underused or overloaded situations on each working node but keep the *vjob* in the *Running* state.

We emphasize that this paper focus on VM context switches as a building block to implement finer scheduling policies. The encapsulation of the vjob into one or several VMs is beyond the scope of the present work and could be addressed later by leveraging solutions such as [16] for instance.

The diagram described in this paragraph shows the different actions that could occur from the scheduler point of view. We define each one of these operations as a VM context switch.

## 2.3 Evaluation of a VM context switch

Evaluation the cost of a cluster-wide context switch, is mandatory since it can significantly degrade the whole performance. According to Figure 1 (c), starting the 4th job can be useless if the suspend time is significant with regards to the running one. In a similar way, resuming the 4th job on distinct resources requires remote accesses which can impact the global performances (locality issues). More generally, migrating, suspending and resuming a VM requires some CPU and memory bandwidth. When there are busy VMs on the nodes concerning by the cluster-wide context switch, it will reduce access to these resources, and thus will take longer to complete the action. All these points have to be analyzed in order to compare the costs of possible actions.

Evaluations have been done on a cluster composed of 11 homogeneous nodes composing of a 2,1 GHz Intel Core 2 Duo (1 CPU composed of 2 cores), 4 GB of RAM and interconnected through a giga ethernet network. Each node runs a linux 2.6.26-amd64 with xen 3.2 and 512 MB of RAM is allocated to the Domain 0. Three NFS storage servers provides the virtual disks for all VMs (debian lenny domUs).

The purpose of these first experiments consists in evaluating the cost of each action. In that sense, we disabled the SMP to have rigorous conditions and launched two VMs on our test node. The first one has 1 GB of memory and aims at stressing the CPU during each experiment. The second one is exploited to evaluate the cost of each transition. The amount of memory allocated to this second VM varies from 512 MB to 2GB. We measured both the duration of each potential context switch operation and the performance loss on the busy VM.

Figures 3(a), 3(b) and 3(c) show the average duration of each action in terms of the memory amount allocated to the manipulated VM. As expected, we observe that the duration of a start/run or stop/shutdown is independent from the VM memory size: booting a VM takes around 6 seconds in our architecture whereas a clean shutdown is longer with approximately 25 seconds (due to the different service timeouts). This second time can be easily reduced by using a ”hard” shutdown of the VM. Concerning migration, suspend and resume operations, first, we see that they clearly depend on the amount of memory allocated to the manipulated VM. Moreover, the way of suspend/resume actions are performed, impact the performance. We conducted several benchmarks to evaluate the cost implied by a local vs. a remote suspend/resume operations (i.e the suspend is done locally and a `scp` or a `rsync` is exploited to push the file somewhere else and reciprocally for the resume). The difference between a local vs a remote resume/suspend is quite significant (twice the duration). These results show the importance of the locality issue in such operations.

Last but not the least, the deceleration factor on the busy VM depends of the duration of the operation[1]. From Figure 3(b) and Figure 3(c), we see that a suspend or a resume operation is proportionally shorter when the memory size of the VM increases. Thus, the deceleration impact is proportionally weaker on the active VM. The average is around 1.5 for `scp` or `rsync` and a bit lesser, 1.3, for local approach. In other word, the impact reaches a maximum of 50% during the transition.

More generally, this study shows the cost of each transition. All these results have to be carefully consider to reduce as many as possible both the global cost and the degradation on running VMs when a cluster-wide context switch occurs.

# 3 Global Design

Cluster-wide context switch mechanisms are implemented in Entropy [5] that already provide the separation between the scheduling strategy and the mechanisms to perform the changes. In a first section, we describe the architecture of Entropy and how the different modules that compose the environment are used. Second, we describe a sample scheduling strategy that exploits the cluster-wide context switch to extend the possibility of common dynamic consolidation strategies to support critical situations such as an overloaded cluster.

[1]Due to spaces limitations, we do not included resulting charts in the paper, reviewers can access them directly on http://entropy.gforge.inria.fr website.

## 3.1 Architecture of Entropy

A cluster for Virtual Machines from Entropy point of view consists of a set of working nodes that can host VMs, a set of storage nodes to serve the virtual disks of the VMs and a set of service nodes that host services such as the head of the distributed monitoring system and the Entropy service. Figure 4 shows the global design of Entropy.

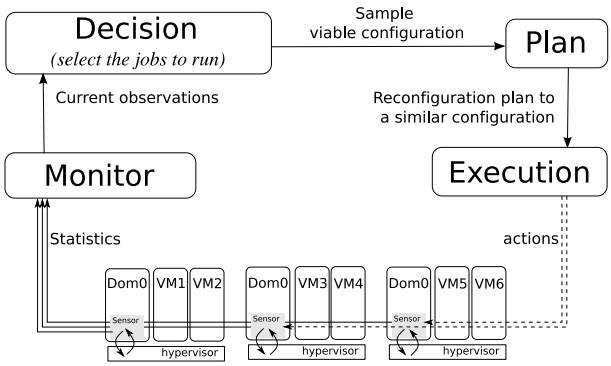



Figure 4: The control loop of Entropy

Entropy acts as a loop that (i) observe the CPU and memory consumptions of the running VMs by requesting an existent monitoring service, (ii) execute an algorithm in the decision module that computes a new solution and indicates the state of the vjobs for the next iteration, (iii) plan the actions to perform the cluster-wide context switch according to the current state of the vjobs and the solution computed in the decision module and (iiii) execute the cluster-wide context switch by performing the actions, implemented with *drivers*. Entropy then accumulates new informations about resource usage, which takes about 10 seconds for our prototype, before repeating the iteration. To reduce the duration of the cluster-wide context switch, the plannification algorithm of Entropy try to compute a viable configuration, similar to the solution computed by the decision module, that require a fewer amount of migrations and fastest resumes.

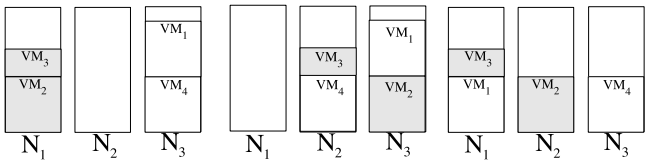
From technical point of view, Entropy currently works with Xen 3.2.1 [17] and Ganglia 3.0.7 [18]. Each of the VMs and each of the Domain-0 are running a Ganglia monitoring daemon. In addition, a shell script running on each Domain-0, provides additional monitoring metrics. Drivers are implemented with SSH commands or use the xen API [19]. The decision module has to be implemented by the administrator to fulfill a specific scheduling policy.

## 3.2 A Sample Decision Module for Dynamic Consolidation

The administrator uses the observations from the monitoring module such as the current resources demand, the state or the assignment of the VMs and combines them with a custom scheduling algorithm using common approaches such as vjob weights or priority queues.

The algorithm in the decision module is responsible of computing a new viable configuration which indicates the state of the vjobs for the next iteration. A viable configuration is a mapping of Virtual Machines (VMs), to nodes that gives every running VMs access to sufficient memory and CPU resources. Figure 5(a) shows a non-viable because the two VMs in gray require their own processing unit while their hosting node

has only 1 CPU. On the other hand, the two configurations in Figure 5(b) are viable because each VM has access to sufficient memory and each node hosts at most one busy VM. We do not consider the waiting and the sleeping vjobs as they do not have any impact on memory or CPU resources. The problem of finding a viable configuration is comparable to the NP-Hard *2-Dimensional Bin Packing Problem* [20], where the dimensions correspond to the amount of memory and the capacity of the processing units.




(a) Non-viable (b) 2 Viable configurations

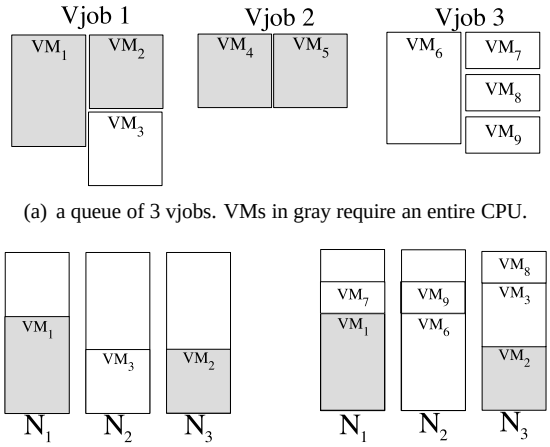
Figure 5: Sample configurations with 3 uniprocessor nodes and 3 VMs. $\texttt{VM}_2$ and $\texttt{VM}_3$ (in gray) require an entire CPU

As an example, we develop an algorithm based on a *First Come/First Serve* (FCFS) policy that provides dynamic consolidation. Every 30 seconds, the algorithm observes the current resources demands of each of the VMs and uses the queue to select the maximum number of vjobs that can run on the cluster. We refer to the problem considered in this phase as the *Running Job Selection Problem*(RJSP).

To select the vjobs to run, we refer to the queue provided by FCFS policy. This queue is ordered in a descending priority order. As the resources demands change over the time for running VMs, some vjobs that were previously sleeping has to be re-evaluated if some resources have been freed. As a consequence the whole queue has to be considered when the new configuration is computed. For each vjob in the queue, a temporary configuration is created and uses the First Fit Decrease (FFD) heuristic to assign the vjob. This heuristic sorts the VMs in a decreasing order regarding to their memory and their CPU demands and try to assign each VM on the first node with a sufficient amount of free resources. If it exists a host for each VM, the current vjob will be considered as running. Otherwise, it will be considered as sleeping (if it is currently sleeping or running) or waiting. After the last iteration over the queue, the list of the vjobs that can run on the cluster is defined and the new configuration is created.

Figure 6 describes the algorithm with a queue of 3 vjobs that have to fit on 3 nodes. Vjob 1 and 2 are currently running while the vjob 3 is waiting. At the first iteration, the algorithm succeeds at computing a viable configuration that can run the VMs of the vjob 1 (see Figure 6(b)). On the second iteration, the algorithm tries to add the vjob 2 and fails as there is no enough free processing units. As the vjob 2 is currently running, its VMs will be move to the sleeping state. Finally, there is a sufficient amount of resources to compute a viable configuration with the vjob 1 and the vjob 3 in the running state (see Figure 6(c)). Thus, for this example, vjob 1 and vjob 3 will be running while the vjob 2 will be suspended during the clusterwide context-switch.

In this situation, the cluster-wide context switch provides the ability to extend traditional dynamic consolidation algorithms that can not handle overloaded clusters. Indeed, in a such environment, it is possible to fix overloaded nodes and maximize the resource utilization by switching the state of the vjobs according to the solution computed by the decision module.




(a) a queue of 3 vjobs. VMs in gray require an entire CPU.

(b) All the VMs of vjob 1 can run

(c) VMs of vjob 3 can run with VMs of vjob 1

Figure 6: Sample construction for the RJSP with 3 vjobs and 3 uniprocessor nodes. VMs in gray require an entire CPU

# 4 Performing a Cluster-wide Context Switch

A cluster-wide context switch consists to switch from the current configuration to a new viable one computed by the decision module. In the rest of this section, we describe the necessary steps to achieve the context switch with a high level of parallelism. Then we estimate its cost. Finally, we explain how to compute a cluster-wide context switch with a cost as reduced as possible using a Constraint Programming approach.

## 4.1 Planning the Cluster-wide Context Switch

The constraint of viability has to be taken into account not only in the final configuration but also during each temporary configuration created during the cluster-wide context switch. In this way, at any time, we can only perform actions that are *feasible*. The actions that suspend and stop a VM liberate resources on its hosting node. The actions that resume and run a VM require resources on their destination node. Finally, the action that migrates a VM liberates resources on its hosting node and requires resources on its destination node. Contrary to suspend and stop, the actions migrate, resume and run are not always feasible and may require to perform actions that liberate resources in prior. Thus it is necessary to plan the actions to ensure that they will be executed when they are feasible.

We identify and resolve two types of plannification issues. First, we ensure the feasibility of each action by ordering them, solving both the *sequential* and the *inter-dependant* constraints. Second, we maintain consistency between inter-dependant VMs (*i.e.* VMs that belong to the same vjob) to allow suspends and resumes of vjobs.

A reconfiguration graph is an oriented multigraph where each edge denotes an action on a VM between two nodes. Each edge specifies the action, the demand of memory $d_m$ and the demand of CPU resources $d_c$. Each node denotes a node of the cluster, with its memory capacity $c_m$ and its CPU capacity $c_c$. Using this representation, it is possible to identify and solve the sequential and the inter-dependant constraints.

A sequential constraint occurs when an action requiring resources can only begin when some actions that liberate resource has completed. In the example in Figure 7, two actions have to be planned: $suspend(\texttt{VM}_2)$ and $migrate(\texttt{VM}_1)$. However these two actions cannot happen in any order or in parallel, because as long as $\texttt{VM}_2$ is on $N_2$, it consumes to much memory to host $\texttt{VM}_1$. Thus, the migration of $\texttt{VM}_1$ can only begin once the suspend of $\texttt{VM}_2$ has completed.

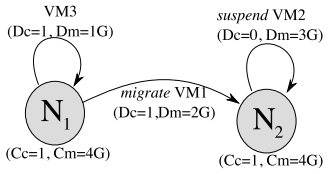
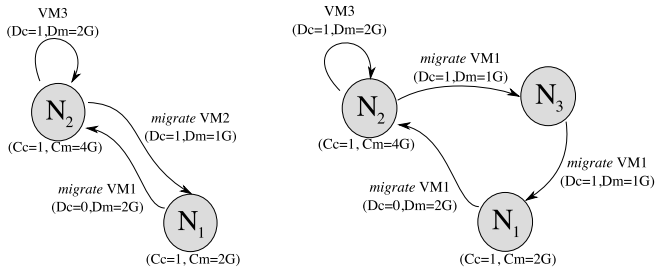



Figure 7: A sequence of actions

An inter-dependant constraint occurs when a set of non-feasible migrations forms a cycle. An example is shown in Figure 8(a), where, due to memory constraints, $\texttt{VM}_1$ can only migrate from node $N_1$ to node $N_2$ when $\texttt{VM}_2$ has migrated from node $N_2$, and $\texttt{VM}_2$ can only migrate from node $N_2$ to node $N_1$ when $\texttt{VM}_1$ has migrated from node $N_1$. We break such a cycle by inserting an additional migration. A *pivot* node outside the cycle is chosen to temporarily host one or more of the VMs. In Figure 8(b), the cycle between $\texttt{VM}_1$ and $\texttt{VM}_2$ is broken by migrating $\texttt{VM}_1$ to the node $N_3$, which is used as a pivot. After breaking the cycle, an order can be chosen for the actions as in the previous example. These actions include moving the VM on the pivot nodes to its original destination.




(a) Inter-dependant migrations

(b) A bypass migration breaks the cycle

Figure 8: Cycle of non-feasible migration

A reconfiguration plan is a solution for the sequencing and the inter-dependant constraints. It ensures the feasibility of each action. To reduce the duration of the cluster-wide context switch and to increase reactivity, it is a necessary to perform as many actions in parallel as possible, so that each action will take place in the minimum possible delay. The plan is composed of a sequence of *pools*, *i.e.* a set of actions. Pools are executed sequentially, where the actions composing them are feasible in parallel.

The reconfiguration plan is created iteratively starting from a reconfiguration graph between the current configuration and the destination configuration. In a first step, we select all the actions that are directly feasible and group them into a pool. If there is no feasible actions, it is necessarily due to an inter-dependant issue. In this situation, we identify a cycle and break it with a bypass migration to create at least one feasible action and add it to the current pool. Then the pool is appended to the plan and a new

reconfiguration graph is created using the temporary resulting configuration of the plan and the configuration we expect. We repeat this step until the resulting configuration of the plan equals the expected configuration.

Figure 9 describes a reconfiguration graph with 4 actions. The associated reconfiguration plan consists of 2 different pools of actions. The first pool executes in parallel the actions $suspend(\texttt{VM}_3)$ and $migrate(\texttt{VM}_1)$ then the second pool executes the actions $resume(\texttt{VM}_5)$ and $run(\texttt{VM}_6)$.

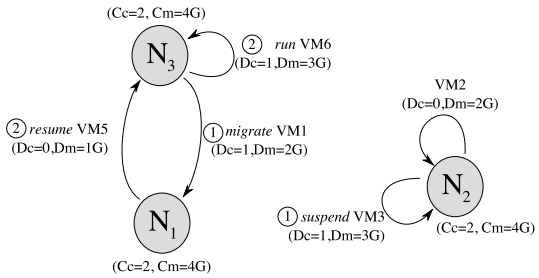



Figure 9: A Reconfiguration Graph

Algorithm in the decision module computes a configuration where all the VMs belonging to the same vjob are in the same state. However, during the cluster-wide context switch, the state may not be consistent for a moment as actions are based on a VM granularity. This may lead to timeouts or failures in the distributed applications running into the VMs. However, experiments show that applications are keep running when the suspend and the resume of the VMs that compose a same vjob is made in parallel and at the exact same time, always in the same order and in a short period [10]. These constraints are not satisfied by the construction of the reconfiguration plan as it consider each VM independently.

A solution to maintain the consistency between the VMs belonging to a same vjob is to alter the plan by grouping the resume and the suspend actions into a same pool to execute them in a short period. The suspend actions are naturally grouped in the first pool as they are always feasible while the resume actions are moved in the pool that initially contains the last resume action. Finally, the actions are sorted using the hostname of the VMs and are pipelined: Each action is started one second after the previous one. It ensures that the VMs are paused sequentially but makes a large part of the writing process in parallel to reduce the duration of these sequences.

## 4.2 A Cost Function to evaluate a Cluster-wide Context Switch

The cost function estimates the cost of a reconfiguration plan. It is model as follow: the cost of a whole plan equals the sum of the total costs of all the actions in the plan. The total cost of an action is the sum of the costs of the preceding pools, plus the local cost of the action. Finally, the cost of a pool corresponds to the cost of the most expensive action in this pool. This model conservatively assumes that delayed an action degrades the cluster-wide context switch.

The local costs are described in the Table 1 using the conclusion of the Section 2.3 where we estimate the cost of each action composing a cluster-wide context switch. We have shown that the cost of the stop and the run action is not lead by the resources demand of the VM but mostly by the software running into it. In this study, we consider

the cost of the run and the stop actions as a constant, arbitrary set to 0. In addition, we have shown that the duration of the suspend action and the migration action is mostly lead by the memory demand of the VM. For those two actions, the cost is set to the amount of memory demand of the VM to manipulate. Finally, the duration of the resume action depends on the memory demand of the VM and its destination location: A *local* resume consists of restoring the state of a VM previously suspended on that node while a *remote* resume requires to move the state file on to destination node first. In this situation, the cost of performing a remote resume is higher than the cost of a local resume.

| Action | Cost |
|---|---|
| migrate($vm_j$) | $\mathcal{D}_m(v_j)$ |
| run($vm_j$) | *constant* |
| stop($vm_j$) | *constant* |
| suspend($vm_j$) | $\mathcal{D}_m(v_j)$ |
| resume($vm_j$) | $\mathcal{D}_m(v_j)$ if *local*,<br>$2 \times \mathcal{D}_m(v_j)$ otherwise |

Table 1: Cost of an action on the VM $v_j$. $\mathcal{D}_m(v_j)$ denotes the memory demand of the VM $v_j$

## 4.3 Optimizing of the Cluster-wide Context Switch

Computing a context switch with a reduced cost using our model leads (i) to perform actions as earlier as possible, (ii) to maximize the size of the pools and (iii) to avoid migrations and remote resumes if possible. These will reduce the duration of the reconfiguration plan, minimize the impact of the reconfiguration process on the environment and on performance. As illustrated in the section 3.2, there are several viable configurations that with the same state for all of the VMs. However, they differ from their reconfiguration plan. Thus, a solution to reduce the cost of the context switch is to compute a viable configuration with an associated cost as reduced as possible. In Entropy, this optimization is made using Constraint Programming (CP).

The idea of CP is to define a problem by stating constraints (logical relations) that must be satisfied by the solution. A *Constraint Satisfaction Problem* (CSP) is defined as a set of variables, a set of domains representing the set of possible values for each variable and a set of constraints that represent required relations between the values of the variables. A solution for a CSP is a variable assignment (a value for each variable) that simultaneously satisfies the constraints. To compute a solution, a constraint solver perform an exhaustive search, based on a depth first search. Entropy uses the Choco library 1.2.04 [21], which can solve a CSPs and optimization problems where the goal is to minimize or maximize the value of a single variable.

To model the assignment of the VM as a CSP, we consider a set of nodes $N$ and a set of VMs $V$. The following vectors describe how to express the state and the assignment of each VM.

**Definition 4.1** *For each node $n_i \in N$, the bit vector $H_i = \langle h_{i1}, \ldots, h_{ij} \rangle$ denotes the set of VMs running on the node $n_i$ (*i.e. *, $h_{ij} = 1$ iff the node $n_i$ is hosting the VM $v_j$). The bit vector $Rdy = \langle r_1, \ldots, r_j \rangle$ denotes the set of VMs that are ready (*i.e.

$r_j = 1$ *indicates that the VM* $v_j$ *is in state sleeping or waiting, depending on its current configuration).*

Using these vectors, we define three elementary constraints: $keepVMState(v_j)$ ensures that the state of a VM $v_j$ must be identical to its current state, while $mustBeRunning(v_j)$ and $mustBeReady(v_j)$ respectively ensure that a VM $v_j$ will be in the Running state in the resulting configuration and in the Ready state. We express the constraints for viable configuration as follows: Let $\mathcal{D}_c$ be the vector of CPU demand of each VM, $\mathcal{C}_c$ be the vector of processing unit capacity associated with each node, $\mathcal{D}_m$ be the vector of memory demand of each VM, and $\mathcal{C}_m$ be the vector of memory capacity associated with each node. The following inequalities express this constraint:

$$\begin{array}{ll} \mathcal{D}_c \cdot H_n \leq \mathcal{C}_c(n_i) & \forall n_i \in N \\ \mathcal{D}_m \cdot H_n \leq \mathcal{C}_m(n_i) & \forall n_i \in N \end{array}$$

These two constraints dynamically evaluate the remaining free place (in terms of both processing unit and memory availability) on each node. This is done in Entropy by solving a ***Multiple Knapsack*** problem using a dynamic programming approach [22]. Thus a solution to the assignment problem is a solution that satisfy both the $multi - knapsack$ constraint and the constraint $keepVMState$ for each VM.

Entropy dynamically estimates the cost of the plan associated with the configuration being constructed based on informations about the VMs that have already been assigned to a node. Then, it estimates a minimum cost for the complete future reconfiguration plan. Finally, the solver determines whether the future configuration based on this partial assignment might improve the solution or will necessarily be worse. In the latter situation, the solver abandons the configuration currently being constructed and searches for another assignment. In principle, the constraint solver must enumerate each possible configuration, check whether it is viable and stop after computing the first solution. In practice, this approach is unnecessarily expensive. Our implementation reduces the computation cost using a number of optimizations. The solver incrementally checks the viability of a configuration as it is being constructed and it discards a partial configuration as soon as it is found to be non-viable. This strategy reduces the number of configurations that must be considered. Choco furthermore tries to detect non-viable configurations by using a ***first fail*** approach [23] in which VMs with important CPU and memory requirements are treated earlier than VMs with lesser requirements. This strategy reduces the chance of computing an almost complete configuration and then finding that the remaining VMs cannot be placed successfully. Moreover, by trying to assign each running VMs on there initial location in priority, the solution tends to reduce the movements of the VMs faster.

# 5 Evaluation

In this section, we evaluate first the scalability of the cluster-wide context switch in Entropy and the ability to reduce its duration. Second we evaluate our proposal with our sample decision module on a cluster composed of 11 working nodes executing vjobs running NAS Grid Benchmarks [24]

## 5.1 Experiments using Workload Traces

We estimate in this theoretical evaluation the scalability of our implementation of the cluster-wide context switch. We compare the resulting reconfiguration plan with a standard heuristic based on a FFD.

These evaluations are based on a set of generated configurations with 200 working nodes, with 2 CPU and 4 GB of memory each, and a variable amount of VMs. For each amount of VMs, 30 different samples are generated. The configurations are build by aggregating several vjobs with specific workloads that correspond to 81 real traces observable on the different benchmarks of the NAS Grid Benchmarks suite for the sizes W, A and B. Each vjob uses 9 or 18 VMs, its initial state is choosed randomly and its assignment satisfies the memory requirement of all the VMs. Each VM requires 256 MB, 512 MB, 1024 MB or 2048 MB of memory and an entire processing unit if it is supposed to execute a computation.

The dedicated node used for these experiments is a Macbook with a Intel Core Duo 1.83 GHz CPU[2] and 2 GB or RAM. The Java heap is increased to 1 GB.

Figure 10 quantifies the reduction of the reconfiguration cost using Entropy. For each configuration, we compare the cost of the reconfiguration plan computed by the FFD in one part and with the cost of the plan computed by Entropy in the other part. The maximum amount of time used by Entropy to reduce the cost of the cluster-wide context switch is set to 40 seconds. We observe that the reconfiguration cost is reduced by an average of 95% using Entropy and the gap becomes more important as the number of possible movements increases with the number of VMs in the configuration. The difference in the cost of the solutions is due to the heuristic that stops after the first completed viable configuration while Entropy keeps computing configurations with a reduced cost until it proves that the cost of the plan is minimum or hits the timeout.

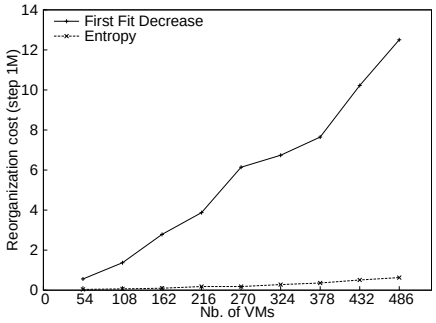



Figure 10: Reconfiguration costs for configurations with 200 nodes.

## 5.2 Experiments on a Cluster

The experimental architecture is the same as in Section 2.3. The experiment consists in running 8 vjobs, each composed of 9 VMs. Vjobs are submitted at the same moment, in a specific order. Each vjob is running an application composed of NASGrid Tasks. The application embedded in the vjob is launched when all the VMs of the vjob are in the Running state. When the application is terminated it signal to Entropy to stop its vjob. Each VM requires a fixed amount of memory, from 512 MB to 2048 MB and

[2]only one core is used by Entropy

requires an entire processing unit when the NASGrid task is executing a computation on the VM.

Figure 11 shows the cost of the cluster-wide context switches performed during the experiment and their durations. Cluster-wide context switches with a small cost and duration only perform migrations, run or stop actions. As an exemple, the five with a cost equals to 0 only perform run and stop actions and takes at most 13 seconds to be performed, the cluster-wide context switch with a cost equals to 1024 performs 3 migrations in 19 seconds. Cluster-wide context switches with a higher cost and duration perform in addition suspend and resume actions. As an example, the one with a cost equals to 4608 takes 5 min 15 seconds to execute 9 stop actions, 18 run actions, 9 resume actions and 9 migrations. This difference in the duration between the cluster-wide context switches is explained by the preliminary study in section 2.3. We shown that the duration of a suspend or a resume action is much long longer that the duration of a migration, a stop or a run action. Finally, we observe that the cost function is a viable solution to avoid to move VMs with migrations or remote resumes if possible: 21 over the 28 resume actions performed during the experiments was made on the nodes that perform the suspend earlier.

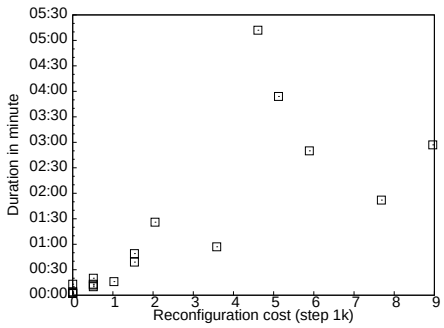



Figure 11: Cost and duration of the 19 cluster-wide context switches performed in the experiment

The second part of this evaluation shows the benefit using a dynamic consolidation combined with a cluster-wide context switch as compared to a traditional static allocation. First, we simulate a FCFS scheduler, the Figure 12 shows its execution diagram. Figure 13(a) and 13(b) show the resources utilization of the VMs with the two different decision modules. The average resource utilization is much more important with in our module until the $30^{th}$ minute. At this moment, resources utilization with Entropy decreases as there is no more vjob to run. At 2 minutes 10, the cluster appears to be overloaded as the running vjobs demand 29 processing units while only 22 are available. In this situation, our decision module computes a new sample configuration and indicates which of all the vjobs must be running to have a viable configuration and the cluster-wide context switch module performs the transition by suspending the vjobs selected by the decision module.

To conclude, exploiting the cluster-wide context switch mechanisms enable to develop scheduling strategy for a fine use resources. Indeed, with the FCFS scheduler, the global completion time is 250 minutes. With the decision module performing dynamic consolidation, the time is reduced to 150 minutes, a reduction of 40%.

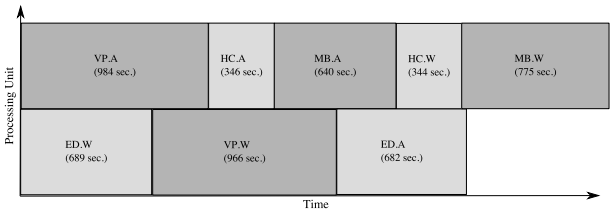
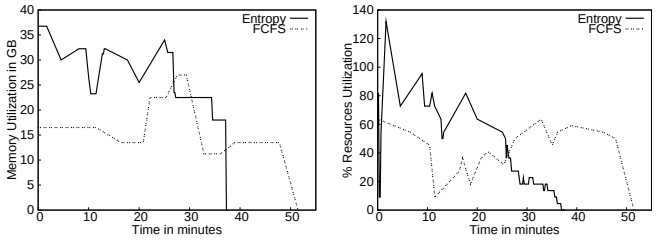



Figure 12: Allocation Diagram for the jobs, with a FCFS Scheduler




(a) Memory Utilization (b) CPU Utilization

Figure 13: Resources utilization of the VMs

# 6 Related Works

Sotomayor *et al.* [25, 4] provide with Haizea the concept of lease as an abstraction for resource provisioning. Users negotiate a amount of resources for a specific duration, indicate if the lease is made in a best-effort mode or use advanced reservations and specify if it is preemptible. Depending on their type, the lease may be migrated, or suspended to free resources for non-preemptible leases or leases with advanced reservations. This approach enables to renew a period of execution for a new amount of time but do not provide a way to dynamically change the set of resources assigned to a lease according to the application needs and the cluster resource changes.

Grit *et al.* [26, 27] consider some VMs replacement issues for resource management policies in the context of Shirako [28] manager. They show the necessity of separating the management policy of the VMs and the mechanisms to perform the changes as we argue in the present work. However, they only consider the VM migration action to perform changes. Even if suspend/resume operations are exploited to resolve sequencing issues, they neglect opportunities provided by such VM transitions.

Fallenbeck *et al.* [11] provide an environment to dynamically adapt the number of slots available for specific scheduling policies with multiple queues. A VM is available on each working node for each queue. Depending on the size of each queue, the amount of corresponding VMs activated varies. This approach reduces the number of idle nodes in clusters as compared to clusters with a static partition scheme of the slots. Our solution is different as we provide a single scheduling environment but improve the resources utilization by acting on the jobs instead of acting on the number of dedicated VMs.

More generally, works addressing the lack of flexibility in resource allocation resolve a particular case without considering a general approach as we describe in this paper.

Finally, several works address the interest of dynamic consolidation in data-centers to provide a efficient use of the resources. Khanna *et al.* [29] and Bobroff *et al.* [8] provide algorithms to minimize the unused portion of resources. However, they do not consider the sequencing and the cyclic issues during the application of the migrations. Wood *et al.* [30] provide a similar environment but in addition, exploit the page sharing between the VMs to improve the packing. They show the interest of planning the changes to detect and avoid sequencing issues but do not consider inter-dependent migrations. In general, all of these solutions provide a algorithm to compute a viable configuration, regarding to some bounded resources specific to their objectives and uses live migrations to perform the change. However, their approaches are limited as they do not consider critical situations such as an overloaded cluster, with no viable assignment to satisfy all the resources requirements. Thanks to the suspend/resume mechanisms provided by the cluster-wide context switches in Entropy, these situations become easily manageable and actions are performed more efficiently due to a finer preparation.

# 7 Conclusion

Most of the clusters use Resources Manager System to schedule the jobs according to users estimates. Recent works point the benefits of a finer scheduling policy by encapsulating jobs into Virtual Machines (VMs) based on their real resources requirements. Each solution differs from its algorithm to manage the jobs but all of them use similar mechanisms to perform the changes. These mechanisms are ad-hoc and cannot easily evolve in order to explore new scheduling policies. In this paper, we have defined the cluster-wide context switch, a generic approach to apply advanced scheduling strategies for resources management based on the life cycle of virtualized jobs (*vjobs*).

The integration in Entropy provides the mechanisms to switch between the different states using live migration, suspend and resume VM capabilities. The construction of the cluster-wide context switch ensures the feasibility of each action while minimizing its whole duration thanks to a cost function and a Constraint Programming approach (CP).

The implementation of a first scheduling algorithm in the decision module shows in our experiments a reduction of the completion time of *vjobs* by 40% as compared to a usual *First Come/First Serve* approach while providing a solution to manage overloaded clusters.

In future works, we plan to provide additional low level relations between the VMs in the decision module. Our approach, based on CP, provides a flexible environment for administrators to specify some constraints such as hosting some VMs on different nodes for high availability considerations. These constraints are already available in Entropy, however they are not maintain during the optimization of the cluster-wide context switch that only consider the state of each VM to compute equivalent configurations. In a second time, we plan to extend the mechanisms provided by the cluster-wide context switch by adding a sleeping state where the VM is simply suspended to ram. This action provides fast resumes and suspend operations that may reduce the duration of the cluster-wide context switch in some situations.

# References

[1] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. E. Sprenkle, "Dynamic virtual clusters in a grid site manager," in *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*. Washington, DC, USA: IEEE Computer Society, 2003, p. 90.

[2] G. Vallee, T. Naughton, and S. L. Scott, "System management software for virtual environments," in *CF '07: Proceedings of the 4th international conference on Computing frontiers*. New York, NY, USA: ACM, 2007, pp. 153–160.

[3] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lantéri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and T. Iréa, "Grid'5000: a large scale and highly reconfigurable experimental grid testbed." *International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 481–494, Nov. 2006.

[4] B. Sotomayor, K. Keahey, and I. Foster, "Combining batch execution and leasing using virtual machines," in *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*. New York, NY, USA: ACM, 2008, pp. 87–96.

[5] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: a consolidation manager for clusters," in *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. New York, NY, USA: ACM, 2009, pp. 41–50.

[6] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI '05)*, Boston, MA, USA, May 2005, pp. 273–286.

[7] P. Ruth, J. Rhee, D. Xu, R. Kennell, and S. Goasguen, "Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure," *Autonomic Computing, 2006. ICAC '06. IEEE International Conference on*, pp. 5–14, 2006.

[8] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, pp. 119–128, May 2007.

[9] A. Verma, P. Ahuja, and A. Neogi, "Power-aware dynamic placement of hpc applications," in *ICS '08: Proceedings of the 22nd annual international conference on Supercomputing*. New York, NY, USA: ACM, 2008, pp. 175–184.

[10] W. Emeneker and D. Stanzione, "Increasing reliability through dynamic virtual clustering," in *High Availability and Performance Computing Workshop*, 2006.

[11] N. Fallenbeck, H.-J. Picht, M. Smith, and B. Freisleben, "Xen and the art of cluster scheduling," in *VTDC '06: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2006, p. 4.

[12] A. W. Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 6, pp. 529–543, 2001.

[13] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the condor experience." *Concurrency - Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[14] P. H. Hargrove and J. C. Duell, "Berkeley lab checkpoint/restart (blcr) for linux clusters," *Journal of Physics: Conference Series*, vol. 46, pp. 494–499, 2006. [Online]. Available: http://stacks.iop.org/1742-6596/46/494

[15] R. Lottiaux, P. Gallard, G. Vallée, C. Morin, and B. Boissinot, "Openmosix, openssi and kerrighed: a comparative study," in *CCGRID 05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid05) - Volume 2*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 1016–1023, iSBN 0-7803-9074-1.

[16] 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 *EuroSys '09: Proceedings of the fourth ACM european conference on Computer systems*. New York, NY, USA: ACM, 2009, pp. 1–12.

[17] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*. Bolton Landing, NY, USA: ACM Press, Oct. 2003, pp. 164–177.

[18] M. Massie, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817–840, July 2004. [Online]. Available: http://dx.doi.org/10.1016/j.parco.2004.04.001

[19] "Xen management api," http://wiki.xensource.com/xenwiki/XenApi, 2008.

[20] P. Shaw, "A constraint for bin packing," in *Principles and Practice of Constraint Programming (CP'04)*, ser. Lecture Notes in Computer Science, vol. 3258. Springer, 2004, pp. 648–662.

[21] N. Jussien, G. Rochart, and X. Lorca, "The CHOCO constraint programming solver," in *CPAIOR'08 workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, Paris, France, Jun. 2008. [Online]. Available: http://www.emn.fr/jussien/publications/jussien-OSSICP08.pdf

[22] M. Trick, "A dynamic programming approach for consistency and propagation for knapsack constraints," in *Proceedings of the Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR-01)*, 2001, pp. 113–124.

[23] R. Haralick and G. Elliott, "Increasing tree search efficiency for constraint satisfaction problems," *Artificial Intelligence*, vol. 14, no. 3, pp. 263–313, October 1980.

[24] M. Frumkin and R. F. V. der Wijngaart, "NAS grid benchmarks: A tool for grid space exploration," *Cluster Computing*, vol. 5, no. 3, pp. 247–255, 2002.

[25] B. Sotomayor, R. M. Montero, I. M. Llorente, and I. Foster, "Capacity leasing in cloud systems using the opennebula engine," in *Cloud Computing and Applications 2008 (CCA08)*, 2008.

[26] L. Grit, D. Irwin, A. Yumerefendi, and J. Chase, "Virtual machine hosting for networked clusters: Building the foundations for "autonomic" orchestration," in *Virtualization Technology in Distributed Computing, 2006. VTDC 2006. First International Workshop on*, Nov. 2006, pp. 1–8.

[27] L. Grit, D. Irwin, V. Marupadi, and P. Shivam, "Harnessing virtual machine resource control for job management," in *Proceedings of the First International Workshop on Virtualization Technology in Distributed Computing (VTDC)*, Nov. 2007.

[28] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. G. Yocum, "Sharing networked resources with brokered leases," in *ATEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2006, pp. 18–18.

[29] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application performance management in virtualized server environments," *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pp. 373–381, 2006.

[30] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, and M. D. Corner, "Memory buddies: exploiting page sharing for smart colocation in virtualized data centers," in *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. New York, NY, USA: ACM, 2009, pp. 31–40.

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