Cluster-Wide Context Switch of Virtualized Jobs

Fabien Hermenier, Adrien Lebre, Jean-Marc Menaud

Cluster-Wide Context Switch of Virtualized Jobs

Fabien Hermenier, Adrien Lèbre, Jean-Marc Menaud
ASCOLA Research Group
Mines de Nantes, INRIA, LINA UMR 6241
firstname.lastname@mines-nantes.fr

ABSTRACT

Clusters are mostly used through Resources Management Systems (RMS) with a static allocation of resources for a bounded amount of time. Those approaches are known to be insufficient for an efficient use of clusters. To provide a finer RMS, job preemption, migration and dynamic allocation of resources are required. However due to the complexity of developing and using such mechanisms, advanced scheduling strategies have rarely been deployed. This trend is currently evolving thanks to the use of migration and preemption capabilities of Virtual Machines (VMs). However, although the manipulation of jobs composed of VM enables to change the state of the jobs according to the scheduling objective, changing the state and the location of numerous VMs at each decision is tedious and degrades the overall performance. In addition to the scheduling policy implementation, developers have to focus on the feasibility of the actions while executing them in the most efficient way.

In this paper, we argue such an operation is independent from the policy itself and can be addressed through a generic mechanism, the cluster-wide context switch. Thanks to it, developers can implement sophisticated algorithms to schedule jobs without handling the issues related to their manipulations. They only focus on the implementation of their algorithm to select the jobs to run while the cluster-wide context switch system performs the necessary actions to switch from the current to the new situation. As a proof of concept, we evaluate the interest of the cluster-wide context switch through a sample scheduler that executes jobs as early as possible, even partially, regarding to their current resources requirements and their priority.

1. INTRODUCTION

Clusters provide high-performance computing, large storage capacity, and high throughput communication for a wide range of applications. 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 of using Resources Management Systems (RMS) where users request resources for a specific duration according to their estimated needs.

Several works have been proposed to provide more flexibility to administrators and users [5, 17, 25]. However, in most cases, the use of clusters is still based on a reservation scheme where resources are statically assigned to jobs for a bounded amount of time. Such a static allocation of resources based on user estimates and an execution of jobs to completion lead to a coarse-grain exploitation of architectures [10]. In the best case, the allocated time-slot is larger than the estimate and resources are underused. In the worst case, running applications can be withdrawn from their resources which may lead to the loss of all the performed calculations. If one may argue that providing dedicated time-slots per job is required in some situations (e.g. for reproducible experiments), most of the end-users do not really care about such requirements and just want to execute their jobs as early as possible and as long as required.

A well-known approach to improve the resource usage consists in exploiting preemption mechanisms where jobs can be processed, even partially, and suspended according to the scheduler objectives [10]. In this case, the RMS performs the transition 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 transition can be considered as a cluster-wide context switch. Unfortunately due to the development complexity of these mechanisms, only few RMS allow such a level of scheduling [8].

Considering the increasing popularity of virtualization in distributed architectures, several works have suggested to exploit VM migration and preemption capabilities to tackle these issues: live migration [6] aims to adapt the assignments of VMs according to their current requirements [3, 15, 23, 28], while the suspend and the resume actions provide preemption [7]. In addition to the scheduling policy implementation, developers have to focus on the feasibility of the actions while executing them in the most efficient way. First, they have to plan the different actions to ensure their feasibility [13, 15, 29]. Second, they have to ensure the correctness of a job when the scheduler suspends or resumes inter-connected VMs. Finally, the duration of a cluster-wide context switch is critical and has to be reduced as possi-

ble: migrating a VM takes up to 26 seconds, while resuming a VM to a distant node takes up to 3 minutes in our experiments. Considering permutation issues is fundamental, however, it does not appear to be a primary concern for the developers of schedulers that should only be focused on efficient algorithms to select the jobs to run while ignoring the mechanisms to perform the transition between the current situation and their solution.

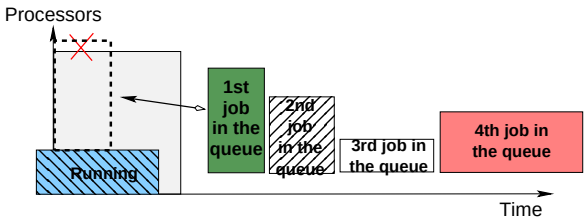
Although some works [9, 25] have stated the lack of flexibility in current resource allocation policies and the interest of using VM capabilities, they did not consider cluster-wide context switch as a fundamental building block. Moreover, they provide ad-hoc solutions to manipulate the VMs without relieving developers of the burden of dealing with permutation issues.

The integration of the cluster-wide context switch module into the consolidation manager Entropy [15] enables to implement more generic scheduling strategies. As a proof of concept, we evaluate a sample scheduling strategy to execute jobs as soon as possible, according to their running priorities and their current resource requirements. When some nodes are overloaded, some VMs are migrated or jobs with the lowest running priority are suspended to give a sufficient amount of resources to jobs with a higher priority. Similarly, when the cluster is considered as underused, jobs that were previously suspended are resumed or new jobs are started. As expected, the use of migration and preemption improve the whole resource usage compared to a static approach.

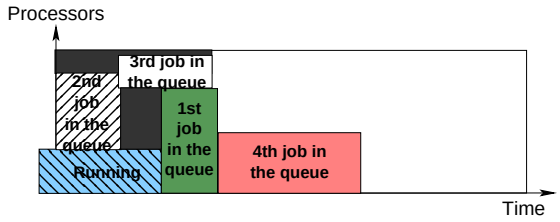
The remainder of the paper is organized as follows: Section 2 expounds current limitations of common batch scheduling approaches. Section 3 introduces the virtualized job abstraction, its life cycle and discusses preliminary experiments about the cost of each action that can occur in a cluster-wide context switch. Section 4 gives an overview of our system and describes the implementation of the cluster-wide context switch. An evaluation of a sample scheduler is presented and discussed in Section 5. Section 6 addresses related work. Finally Section 7 concludes this paper and gives some perspectives.

2. BATCH SCHEDULER LIMITATIONS

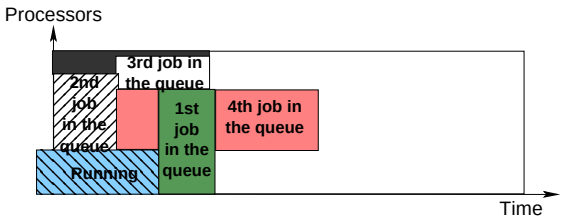
Most of the clusters rely on a reservation scheme where RMSs assign a static set of resources to a job for a bounded amount of time [8]. The common policy consists in scheduling submitted jobs in a *First Come, First Serve* (FCFS) strategy with the EASY backfilling mechanism [19, 24]. Figure 2(a) depicts this process: jobs are enqueued one after the other and are scheduled according to their estimated duration and resource requirements. The backfilling mechanism deals with fragmentation issues while guaranteeing a time reservation for the first job in the queue (see Figure 2(b)). Although there exist more advanced backfilling strategies, such as Conservative which provides time guarantee for each waiting job in the queue, these strategies suffer from limitations that prevent an optimal usage of the resources without exploiting advanced mechanisms such as preemption. Such a functionality enables to run a job, even partially, each time it is possible (see Figure 2(c)).



(a) Jobs 1, 2, 3 and 4 arrive in the queue and have to be scheduled



(b) FCFS + EASY back filling: jobs 2 and 3 have been back filled. Some resources are unused (dark gray areas).



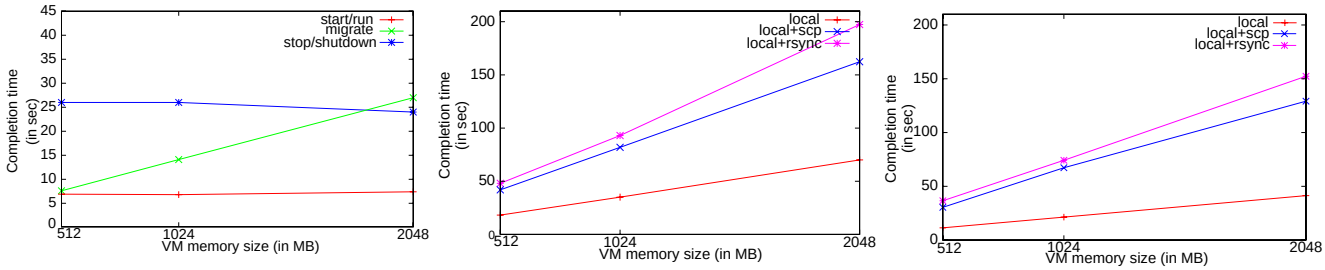
(c) EASY back filling with preemption. The job 4 is started earlier without impacting job 1. A small piece of resources is still unused (dark gray areas). Thanks to the consolidation and preemption mechanisms, it is possible to improve the whole cluster usage.

Figure 2: Backfilling limitations

When submitting jobs, users often provide under or over estimated time and resources requirements [21]. A job may complete before the end of its time slot without referring the RMS that resources can be freed. In addition, the "reservation mode" available in the majority of RMS allows users to book more nodes than required without checking if all resources are really exploited at runtime. Finally, despite batch schedulers try to backfill as soon as one job has been stopped, backfilling strategies cannot manage application requirement changes. In all of these situations, dynamic scheduling is mandatory to finely exploit cluster resources.

Usually used for fault-tolerance issues, checkpointing solutions, like checkpointing-based resource preemption, have been suggested to provide finer scheduling strategies [14, 27]. However, these methods are strongly middleware or OS dependent and they do not consider application resource changes. Single System Images such as openMosix or Kerrighed [20] have integrated advanced strategies based on processes migrations and preemptions. Unfortunately, due to the complexity of the development, most of their implementation have not been finalized.

Virtualization [22] can solve the lack of flexibility in cluster scheduling policies thanks to live migration, suspend and resume capabilities. In this setting, each component of a



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

Figure 1: Duration of each action according to the amount of memory allocated to the VM

job is executed into its own virtual machine (VM) while each working node executes a hypervisor, such as Xen [1], to run and manipulate the VMs.

3. FUNDAMENTS

This section deals with the fundaments of our proposal. First, we define the *virtualized job* abstraction and the different actions that change its state. Second, we evaluate the cost of each action.

3.1 Virtualized job

We reconsider the batch scheduler granularity from the usual job to the virtualized one and define the *virtualized job* (*vjob*) abstraction. According to the nature of the application to execute (centralized or distributed), a *vjob* can be composed of several VMs. Implementing dynamic scheduling policies consists in manipulating vjobs through their four different states described in Figure 3: `waiting`, `sleeping`, `running` or `terminated`. The transition between the different states consists in applying the associated action to all the VMs composing the vjob. Actions are supposed to be atomic: all the VMs composing a vjob are in the same state. The action *migrate* slightly differs from others. It relocates a VM from one node to another with a negligible downtime using live migration [6]. This action can be performed on subsets of vjobs as it does not change the state of the manipulated VMs.

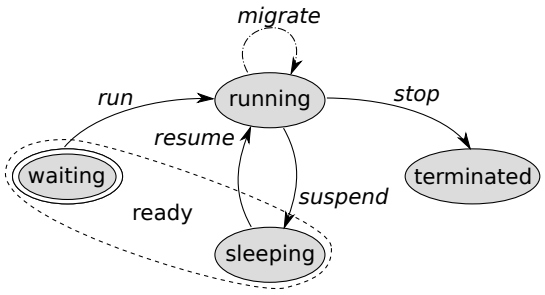


Figure 3: The life cycle of a virtualized job. Each transition but migrate implies to execute the associated action on each VM attached to the vjob

In details, the action *run* starts the selected vjob and puts it into the state `running`. It consists in launching the VMs of the vjob on the cluster. A vjob can be suspended on a persistent device to free resources with the action *suspend*. This lets the vjob into the state `sleeping`. Reciprocally, a suspended vjob can be resumed with the action *resume*: each VM is resumed from its state file previously saved. Finally, the action *stop* shuts down a vjob and puts it into the state `terminated`. The pseudo state `ready` combines the states `waiting` and `sleeping`. Both precede the state `running` and if the actions to perform the transition are different, these states are equivalent regarding to the availability of the vjob and its resources consumption.

Relying on this model, implementing a dynamic cluster scheduler consists of the following steps, (i) determine the set of vjobs to run, (ii) associate a sufficient amount of resource to each VM attached to the selected vjobs, and (iii) execute the mandatory actions to perform the vjobs transition.

The main contribution of our proposal is to provide a generic system to deal with the two latest steps. We refer to this as a *cluster-wide context switch*.

3.2 Evaluation of a VM context switch

Evaluating the cost of a cluster-wide context switch is mandatory since it significantly reduces the performance on the nodes involved in the action. Indeed migrating, suspending or resuming a VM requires some CPU resources and memory bandwidth. When the involved nodes host CPU intensive VMs, performing the action reduces their access to these resources for the whole duration of the action. The purpose of these first experiments consists in evaluating the duration of each action.

Evaluations have been done on a cluster composed of 11 homogeneous nodes including a 2.1 GHz Intel Core 2 Duo, 4 GB of RAM and interconnected through a giga ethernet network. Each node runs a GNU/Linux 2.6.26-amd64 with Xen 3.2 and 512 MB of RAM are allocated to the Domain-0. Three NFS storage servers provide the virtual disks for all the VMs.

Figures 1(a), 1(b) and 1(c) show the average duration of each action depending on the amount of memory allocated to the manipulated VM. As expected, the duration of a run or a stop action is independent from the amount of memory allocated to the VM: booting a VM takes around 6 seconds in our architecture whereas a clean shutdown takes approximately 25 seconds. This second duration is due to the different service timeouts and can be easily reduced to a second by using a "hard" shutdown of the VM. On the opposite, the

duration of a migration, a suspend or a resume action on a VM clearly depends on the amount of memory allocated to it. Moreover the duration of a suspend or a resume action depends on the involved nodes. We conducted several benchmarks to evaluate the cost implied by a local or a remote suspend/resume operation (*i.e.* the suspend is done locally and the command `scp` or `rsync` pushes the file on the destination node and reciprocally for the resume). The difference between a local and a remote resume/suspend is quite significant (twice the duration). These results show the interests of preferring local suspend and resume actions instead of the remote ones to reduce the duration of a cluster-wide context switch.

4. GLOBAL DESIGN

As a proof of concept, we chose to extend the consolidation manager Entropy [15] with the cluster-wide context switch mechanism. Entropy focused on hosting all the running VMs on the minimum number of nodes using migrations. Since it did not consider the whole life cycle of the VMs, it was not able to solve overloaded situations that require to suspend VMs. In addition, it manipulated each VM individually and was not able to ensure a consistent state for a set of interconnected VMs. After describing the changes made to Entropy, we present our implementation of the cluster-wide context switch.

4.1 Architecture

We consider a cluster as 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. A *configuration* describes the CPU and the memory capacity of each node and the CPU, the memory requirement and the position of each VM in the cluster. A viable configuration denotes a configuration where each running VM has an access to a sufficient amount of memory and CPU resources to satisfy its requirements. Our modifications on Entropy make it able to implement the cluster-wide context switch.

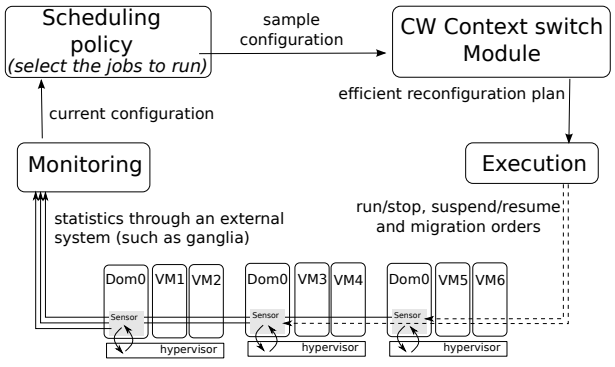


Figure 4: The control loop of Entropy

The system acts as a loop (see Figure 4) that (i) extracts the current configuration from a monitoring service, (ii) executes the scheduling strategy defined by administrators that indicates the state of the vjobs for the next iteration, (iii) determines from the current and the new configuration the actions to perform and plans them and (iiii) executes the cluster-wide context switch by performing the actions.

4.2 Implementation

Performing a cluster-wide context switch requires to execute several actions on VMs in a correct order and an efficient way. In this section, we first address the feasibility of these actions. Second we describe our solution to ensure the correctness of suspending and resuming a set of interconnected VMs. Finally, we present our approach to reduce the duration of its execution.

4.2.1 Plan the actions on VMs

Regarding to the life cycle of the vjobs, their current state and their state computed by the scheduling policy for the next round, we determine the actions to apply on VMs. These actions must be planned to ensure their feasibility. Some actions have to be executed before others [13, 29] and additional migrations have to be considered to solve inter-dependency issues [15]. Solving these dependencies consists in providing a *reconfiguration plan* that describes an execution protocol ensuring the feasibility of the actions. A reconfiguration plan is modelled as a sequence of *steps*, *i.e.* a set of actions that are executed sequentially, while the actions composing them are executed in parallel. To reduce the duration of the cluster-wide context switch and to increase reactivity, it is mandatory to perform in parallel as many actions as possible so that each action takes place with the minimum possible delay.

The reconfiguration plan is created iteratively from a reconfiguration graph that describes the actions to perform to pass from one configuration to another. A reconfiguration graph is an oriented multigraph where each node denotes a machine and each edge denotes an action on a VM between two machines. First, all the feasible actions are grouped into a step. If there are no feasible actions, it is due to an inter-dependent issue. In this situation, the cycle is broken with an additional migration on a temporary node to create at least one feasible action, added to the current step. Then the step is appended to the plan, and a new reconfiguration graph is created using the temporary resulting configuration of the plan and the expected configuration. This process is repeated until the resulting configuration of the plan equals the expected configuration.

4.2.2 Suspending and resuming a vjob

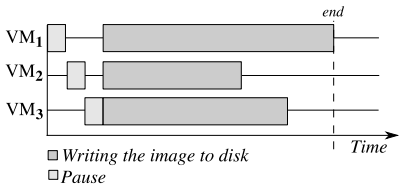
When several VMs execute a distributed application, they exchange network packets through a transport protocol, commonly TCP. This protocol maintains a total ordering between IP packets and ensures their delivery. When a host does not receive an acknowledgment for a packet after multiple retransmissions or when the *keep-alive* timeout is reached, the protocol considers the receiver as unreachable and closes the connection.

During the cluster-wide context-switch, the states of VMs belonging to the same vjob may not be consistent for a moment since actions are based on a VM granularity and some VMs may not be reachable during this time. Thus it is necessary to coordinate the suspend and the resume actions when the cluster-wide context switch manipulates the state

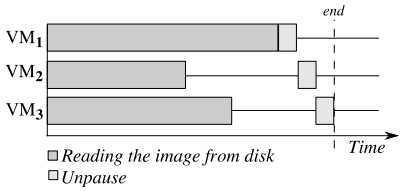
of several VMs belonging to the same vjob. This coordination ensures that the distributed application will not fail during the reconfiguration.

Suspending a VM consists in pausing it then writing the image of its memory to a disk. When the VM is paused, it does not execute any instructions. On the opposite, resuming a VM consists in restoring its memory by reading its image from the disk then unpausing it. While executing a pause (or an unpause) takes only a few milliseconds, reading (or writing) the image from (to) the disk takes a significant amount of times, led by the amount of memory allocated to the VM (see Section 3.2).

Once one of the VMs is paused while others are running, the state of the vjob is not consistent. To avoid network errors, the VMs will have to be in the same state (sleeping or running) before outreaching the *unreachable* delay of TCP. A solution to efficiently coordinate several suspend and resume actions without any clock synchronization is to pause each VM sequentially, in a deterministic order using their unique identifier. To coordinate the suspend actions, all the VMs are first paused sequentially then the content of their memory is written in parallel to the disk (Figure 5(a)). To coordinate several resumes, the images are read in parallel then all the VMs are unpaused sequentially in the same order they were suspended earlier (Figure 5(b)). As the pause and the unpause actions take only a few milliseconds, this solution ensures that the delay between the first and the last pause (or unpause) is inferior to the *unreachability* delay while non-acknowledged packets will be retransmit.



(a) Suspend



(b) Resume

Figure 5: Several suspend and resume actions are coordinated by sequencing pause and unpause actions

Our coordination protocol can be integrated into Entropy by altering the reconfiguration plan. It consists in grouping the resume and the suspend actions into the same step. The suspend actions do not have precondition so they are naturally grouped into the first step as they are always feasible. The conception of a plan ensures that if an action is feasible at a specific step, it is feasible for all the following steps. Thus, to satisfy the dependencies of the resume actions, each action is grouped into the step that initially contains the last resume action. Figure 6 shows an unmodified reconfiguration plan, it suspends a vjob j_1 composed of the VMs vm_1 and vm_2, resumes a vjob j_2 composed of the VMs vm_4 and vm_5, the VMs vm_3 and vm_6 belong to other vjobs that stay in the running state. Figure 7 shows the modified version of this plan with the coordination of the actions related to vjobs j_1 and j_2.

$$\begin{array}{ll} s_1: & suspend(vm_1)\ \&\ suspend(vm_2) \\ s_2: & resume(vm_4)\ \&\ migrate(vm_3) \\ s_3: & resume(vm_5)\ \&\ migrate(vm_6) \end{array}$$

Figure 6: Unmodified reconfiguration plan with three steps s_1, s_2, and s_3. ' & ' indicated the parallelism.

$$\begin{array}{ll} s_1: & pause(vm_1) \\ s_2: & pause(vm_2) \\ s_3: & write(vm_1)\ \&\ write(vm_2) \\ s_4: & migrate(vm_3) \\ s_5: & migrate(vm_6)\ \&\ read(vm_4)\ \&\ read(vm_5) \\ s_6: & unpause(vm_4) \\ s_7: & unpause(vm_5) \end{array}$$

Figure 7: Reconfiguration plan to coordinate the suspend and resume actions of the vjob j_1 and j_2.

4.2.3 Reducing the duration of the execution

To reduce the duration of a cluster-wide context switch, Entropy computes several similar configurations where all the vjobs of each configuration have a state identical to the sample configuration provided by the scheduling algorithm. Indeed, according to the scheduler policy, the location of the VMs is not relevant, only the state of each vjob is significant. However from a cluster-wide context switch point of view, each transition has a duration. As a consequence, we should be able to compare reconfiguration plans with a cost function and select the one with the smallest estimated duration.

The cost of a plan is a positive value. Its has no unit but it allows to compare different plans when they reach equivalent configurations. The cost of a reconfiguration plan P composed of a sequence of n steps s and x actions a is defined through an objective function K described in the Equation (1). This model assumes the cost of an action is minimal when it is executed in the first step. Thus, delaying an action increases the cost of a plan.

$$\begin{aligned} K(p) &= \sum_{x=0}^{m} K(a_x) \\ a_x \in s_i, K(a_x) &= \sum_{j=0}^{i-1} K(s_j) + k(a_x) \\ K(s_i) &= max(k(a_x)), a_x \in s_i \end{aligned} \quad (1)$$

The local cost k of an action a_x relies on the experiments described in the Section 3.2. The different values for $k(a_x)$ are listed in the Table 1. We showed that the duration of both the stop and the run actions is constant and depends on the software running in the involved VM. For this study,

their cost has been set to 0. The duration of a suspend and a migrate actions is led by the memory requirement of the involved VM. We set this cost to the amount of memory allocated to it. Finally, the duration of the resume action depends on the memory demand of the VM and its destination location. Indeed, a *local* resume does not require to move the image file to the destination node. We choose to define the cost of a remote resume as twice the cost of a local one. By such a way, we favor local resume.

Type of a_x	$k(a_x)$
migrate	$\mathcal{D}_m(vm_i)$
run	*constant*
stop	*constant*
suspend	$\mathcal{D}_m(vm_j)$
resume	$\mathcal{D}_m(vm_j)$ if *local*, $2 \times \mathcal{D}_m(vm_j)$ otherwise

Table 1: Local cost of an action a_i. $\mathcal{D}_m(vm_j)$ denotes the memory allocated to the VM vm_j

To compute the different viable configurations, Entropy uses the Choco solver [4] which is based on a Constraint Programming [2] approach.

5. PROOF OF CONCEPT

As a proof of concept, we describe the implementation of a sample scheduler that only indicates at each round the states of the vjobs. According to the changes, the cluster-wide context switch system handles the transition and provides transparent dynamic resources allocation, live migration and vjob preemption. We discuss the execution of this algorithm relying on the cluster-wide context switch on a cluster composed of 11 working nodes running NAS Grid Benchmarks [11].

5.1 A sample scheduler

A common objective of schedulers is to execute jobs as soon as possible. The use of preemption, migration and dynamic allocation of resources improves the cluster usage by executing jobs, even temporarily, when there is a sufficient amount of resource to satisfy their requirements. Our sample algorithm tends to satisfy those principles.

The algorithm computes a list of vjobs, initially empty, that specifies the vjobs that must be running for the next iteration. To avoid starvation, all the vjobs are stored in a queue with a FCFS priority. For each vjob in the queue, we check if it exists a viable configuration composed of the current vjob and the vjobs already in the list. This assumption is made using the First Fit Decrease algorithm that assigns each VM composing the vjob to the first node with a sufficient amount of free resources. If the algorithm succeeds in assigning all the VMs then the vjob is added to the list. After the last iteration over the queue, the list of vjobs that will be in the `running` state is defined, the other vjobs will be in the `ready` state.

5.2 Experiment on a cluster

The experiment consists in scheduling 8 vjobs, each composed of 9 VMs, using our sample scheduling algorithm. Although vjobs are submitted at the same moment, they are enqueued in a deterministic order to ensure the reproducibility of the experiments. Each vjob is running an application composed of NASGrid Tasks. The application embedded in the vjob is launched when all the VMs are in the `running` state. When the application is terminated, it signals to Entropy to stop its vjob. Each VM requires a fixed amount of memory, from 512 MB to 2048 MB and requires an entire CPU when the NASGrid task is executing a computation on the VM. With the Ganglia monitoring system, the monitoring sensor of running on each working node takes at most 10 seconds to signal a change to the monitoring frontend. This reactivity has a negative impact on the ability of Entropy to quickly fix a non-viable configuration. The use of a faster monitoring system should improve this situation.

Computing the reconfiguration plan with the minimum cost may be time consuming. Choco has the property that it can be aborted at any time, in which case it returns the best result computed so far. This makes it possible to impose a time limit on the solver, to ensure the reactivity of the approach. In previous experiments [15], Entropy computes reconfiguration plans with 200 nodes and 400 VMs in one minute. For the current experiment, the total computation time is limited to 30 seconds and Choco computes each plan in less than 20 seconds.

Figure 8 shows the cost and the duration of the cluster-wide context switches performed during the experiment. Those with a small cost and duration only perform migrations, run or stop action. As an example, the five with a cost equal to 0 only perform run and stop actions and take at most 13 seconds. The cluster-wide context switch with a cost equal to 1024 performs 3 migrations in 19 seconds. Cluster-wide context switches with a higher cost perform in addition suspend and resume actions. This increases significantly the duration of their execution. As an example, the one with a cost equal to 4608 takes 5 min 15 seconds to execute 9 stop actions, 18 run actions, 9 resume actions and 9 migrations. In addition, the cost of the plan is not related to its duration when it implies different kind of actions. As mentioned in Section 4.2.3, the cost function only enables to determine the best reconfiguration plan when all the VMs in the destination configuration have the same state. At this moment, our cost function cannot be used to estimate the duration of one cluster-wide context switch. However it appears to be a viable solution to avoid to migrate VMs or perform remote resumes if possible: 21 over the 28 resume actions performed during the experiment were made on the nodes that perform the suspend earlier.

The second part of this evaluation analyses the benefit of the cluster-wide context switch with regards to to a static approach. In theory, the gain provided by dynamic scheduling policies is related to the resources requirements of the vjobs. In the best case, requirements evolve during the execution of the vjob, and the benefits are significant. In the worst case, requirements are constants and a static approach with backfilling is sufficient. As in many cases, the NASGrid benchmarks used in this experiment do not require an entire CPU all the time.

In this experiment, the static scheduling policy relies on a simulated *First Come, First Serve* (FCFS) algorithm. The

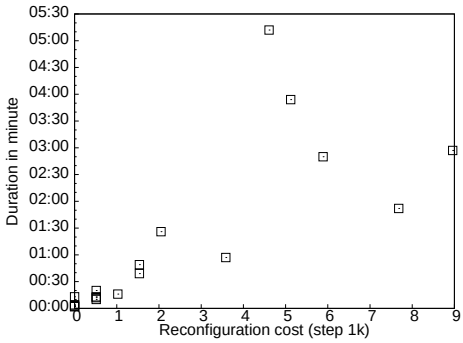


Figure 8: Cost and duration of the 19 cluster-wide context switches performed during the experiment

FCFS scheduler selects the vjobs to execute by iterating over a queue. When there is a sufficient amount of free resources to execute all the VMs composing a vjob, the scheduler allocates a CPU and a sufficient amount of memory to each VM for the whole duration of the vjob. Each vjob is composed of 9 VMs, thus requiring 9 CPUs to run. As each vjob requires the same amount of CPU to run, a backfilling strategy will not improve scheduling (Figure 9 shows the execution diagram of the vjobs).

Figure 10(a) and 10(b) show the resources usage of the VMs with the two different schedulers. The average resources usage is more important with our module during the first 30 minutes. Afterwards, resources utilization with Entropy decreases as there are no more vjobs to run. At 2 minutes 10, the cluster is overloaded as the running vjobs demand 29 processing units while only 22 are available. In this situation, the scheduler indicates which of all the vjobs must be running to have a viable configuration and the cluster-wide context switch performs the transition by suspending the vjobs that must be in the `ready` state. This improvement of the resource usage reduces the cumulated completion time of the vjobs. The cumulated completion time for the 9 vjobs is equals to 250 minutes with the FCFS scheduler. It is reduced to 150 minutes with our sample scheduling policy.

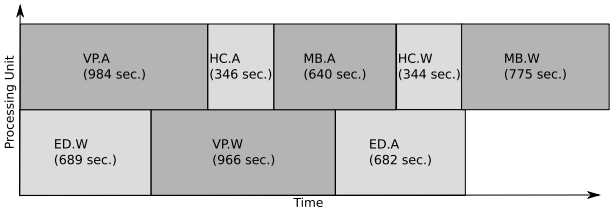
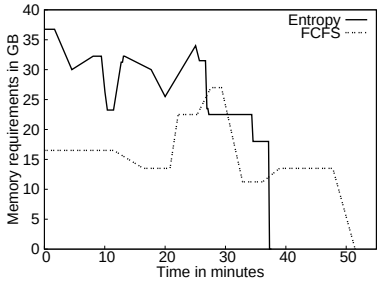


Figure 9: Execution diagram for the vjobs with a FCFS Scheduler

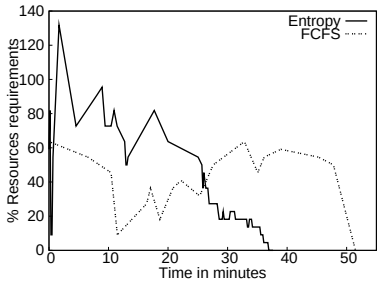
To conclude, one can implement finer scheduling strategies by focusing on the scheduling policy and leaving the issues related to the vjob permutation to the cluster-wide context switch module.

6. RELATED WORK

Sotomayor *et al.* [25, 26] provide with Haizea the concept of lease as an abstraction for resource provisioning. Users negotiate an amount of resource for a specific duration by



(a) Memory requirements



(b) CPU requirements

Figure 10: Resources utilization of the VMs

indicating (i) wether the lease is made in a best-effort mode or uses advanced reservations and (ii) if it is preemptible. Depending on its 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 does not provide a way to dynamically change the set of resources assigned to a lease with variable needs.

Grit *et al.* [12, 13, 16] describe in Shirako the necessity of separating the management policy of the VMs and the mechanisms to perform the changes as we argue in the present work. They consider the sequencing issues when migrating several VMs for resource management policies, However to solve inter-dependencies, they choose to suspend a VM to break the cycle of dependencies instead of using a bypass migration. Regarding to our experimental study, suspending a VM is much more time consuming than the use of a bypass migration. Moreover, they only consider VM migration in their VM manager. By ignoring the possibilities to manipulate the state of the VMs, it is not possible to implement advanced scheduling policies.

Fallenbeck *et al.* [9] 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 activated VMs 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 that acts on the jobs instead of acting on the number of slots per queue.

All of these works address the lack of flexibility in RMSs and use VM mechanisms to improve it. Each solution ad-

dresses a particular use case. All have the same issues when manipulating VMs but all use ad-hoc mechanisms to solve them without considering a general approach as we describe in this paper.

Finally, several works address the interest of dynamic consolidation in datacenters to provide an efficient use of the resources. Khanna *et al.* [18] and Bobroff *et al.* [3] provide algorithms to minimize the unused portion of resources. However, they do not consider the sequencing and the cyclic issues when performing the migrations. Wood *et al.* [29] provide a similar environment and 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 an algorithm to compute a viable configuration, specific to their objectives and use live migrations to perform the changes. 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 resource requirements. Thanks to the suspend/resume mechanisms provided by the cluster-wide context switch in Entropy, these situations become easily manageable and actions are performed more efficiently due to a finer plan.

7. CONCLUSION

For an efficient use of resources in clusters, advanced scheduling algorithms require preemption, migration and dynamic allocation of resources to the jobs. These mechanisms must be considered by developers in addition to the implementation of the algorithm that selects the jobs to run. This increases the complexity of developing advanced algorithms and prevents their deployment on clusters. In this paper, we defined the cluster-wide context switch, a building block leveraging virtualization capabilities to facilitate advanced scheduling implementations. It relies on the manipulation of *virtualized-jobs* (vjobs), *i.e.* jobs composed of VMs, through their life cycle. The cluster-wide context switch is implemented in the consolidation manager Entropy. The developer is only focused on an algorithm to select the vjobs to run. Then the cluster-wide context switch infers and plans the actions to perform on the VMs. It ensures the feasibility of the process and selects, amongst the multiple satisfying solutions, the one implying the fastest change.

As a proof of concept, we evaluated the cluster-wide context switch through a sample dynamic scheduler. Transparently for developers, it uses migration, dynamic allocation of resources and preemption to satisfy scheduler objectives. In future work, we plan to extend our cost function to be able to estimate the duration of a cluster-wide context switch. It requires deeper investigations (with respect to the current ones) first on the cost of each action and second on the side effects of performing them simultaneously.

8. REFERENCES

[1] 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*, pages 164–177, Bolton Landing, NY, USA, Oct. 2003. ACM Press.

[2] F. Benhamou, N. Jussien, and B. O'Sullivan, editors. *Trends in Constraint Programming*. ISTE, London, UK, May 2007.

[3] 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*, pages 119–128, May 2007.

[4] H. Cambazard, N. Jussien, F. Laburthe, and G. Rochart. The choco constraint solver. In *INFORMS Annual meeting*, Pittsburgh, PA, USA, Nov. 2006.

[5] 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)*, page 90, Washington, DC, USA, 2003. IEEE Computer Society.

[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)*, pages 273–286, Boston, MA, USA, May 2005.

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

[8] Y. Etsion and D. Tsafrir. A short survey of commercial cluster batch schedulers. Technical Report 2005-13, The Hebrew University of Jerusalem, may 2005.

[9] 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*, page 4, Washington, DC, USA, 2006. IEEE Computer Society.

[10] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. Theory and practice in parallel job scheduling. In *IPPS '97: Proceedings of the Job Scheduling Strategies for Parallel Processing*, pages 1–34, London, UK, 1997. Springer-Verlag.

[11] M. Frumkin and R. F. V. der Wijngaart. NAS grid benchmarks: A tool for grid space exploration. *Cluster Computing*, 5(3):247–255, 2002.

[12] 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.

[13] 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*, pages 1–8, Nov. 2006.

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

[15] 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, pages 41–50, New York, NY, USA, 2009. ACM.

[16] 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*, pages 18–18, Berkeley, CA, USA, 2006. USENIX Association.

[17] O. Khalid, R. J. Anthony, P. Nilsson, K. Keahey, M. Schulz, K. Parrot, and M. Petridis. Enabling and optimizing pilot jobs using xen based virtual machines for the hpc grid applications. In *VTDC '09: Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*, pages 1–8, New York, NY, USA, 2009. ACM.

[18] 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*, pages 373–381, 2006.

[19] D. A. Lifka. The anl/ibm sp scheduling system. In *IPPS '95: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 295–303, London, UK, 1995. Springer-Verlag.

[20] 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*, pages 1016–1023, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7803-9074-1.

[21] 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.*, 12(6):529–543, 2001.

[22] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, 1974.

[23] 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*, pages 5–14, 2006.

[24] J. Skovira, W. Chan, H. Zhou, and D. A. Lifka. The easy - loadleveler api project. In *IPPS '96: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 41–47, London, UK, 1996. Springer-Verlag.

[25] 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*, pages 87–96, New York, NY, USA, 2008. ACM.

[26] 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.

[27] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[28] 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*, pages 175–184, New York, NY, USA, 2008. ACM.

[29] 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*, pages 31–40, New York, NY, USA, 2009. ACM.