Schlouder: A broker for IaaS clouds

Etienne Michon, Julien Gossa, Stéphane Genaud, Léo Unbekandt, Vincent Kherbache

Schlouder: a broker for IaaS clouds

Étienne Michon[a,*], Julien Gossa[a], Stéphane Genaud[a], Léo Unbekandt[a], Vincent Kherbache[b]

[a] *ICube, University of Strasbourg, CNRS, 300 bd Sébastien Brant - CS 10413, F-67412 Illkirch Cedex, France, +33 3 68 85 45 42*
[b] *INRIA, Sophia Antipolis - 2004, route des Lucioles - 06902 Sophia Antipolis*

Abstract

In the field of cloud computing, Infrastructure as a Service (IaaS) provides virtualized on-demand computing resources on a pay-per-use model. IaaS Cloud differ from traditional mutualized infrastructures in that the resources can be dynamically claimed and released, and the real hardware infrastructure is unknown to its users. These properties drastically changes the way resource provisioning and job scheduling can be addressed by the user because i) the large number of jobs and resources to handle becomes rapidly overwhelming for human operators, and ii) the real performances of the platform should be inferred from observations to make robust scheduling decisions. In order to optimize the resources usage by the client, we advocate the need for brokers on the client-side. This article presents our work based on *Schlouder*, a broker of IaaS cloud resources able to provision and schedule independent jobs or static workflows according to strategies chosen by the client. Further, we advocate that simulation can be a precious auxiliary to help the user to choose between provisioning strategies. Schlouder brings a unique feature which is to predict through simulation the makespan and cost of executions under various strategies. The contribution of this work is twofold. First, it presents the broker, available as an open source project, in which new provisioning strategies can be plugged in by third parties. The effectiveness of the tool is demonstrated through experiments involving actual applications and platforms. Second, we show that simulation produces accurate predictions making this feature a helpful means for the user to choose the appropriate strategy.

Keywords: IaaS, Cloud Broker, Schlouder, Provisioning, Simulation.

*Corresponding author

Email addresses: `etienne.michon@etu.unistra.fr` (Étienne Michon), `julien.gossa@unistra.fr` (Julien Gossa), `genaud@unistra.fr` (Stéphane Genaud), `leo@unbekandt.eu` (Léo Unbekandt), `vincent.kherbache@inria.fr` (Vincent Kherbache)

1. Introduction

The need for computational and storage resources has been constantly increasing during the last decades, and today, *cloud computing* represents an attractive solution for a wide variety of users. Cloud computing can be delivered to the customers under different forms, IaaS, Platform as a Service (PaaS) or Software as a Service (SaaS) to name a few. We focus in this paper on the IaaS paradigm, which enables the user to use the bare infrastructure through the deployment of virtual machines (VMs). This form is the most generic one as almost any software can be installed and run. Although clouds are often tightly associated with web applications, IaaS enables a wide range of applications, from media encoding and decoding to remote software delivery and data mining. Previous works also demonstrate the possibility to use the cloud as a viable infrastructure for scientific computations. Montero et al. [1], Song et al. [2] and Villegas et al. [3] showed the usefulness of the cloud for different type of scientific application such as bag-of-tasks, workflows, and MPI programs.

Managing the resources provided by an IaaS implies however two major difficulties from the user perspective: i) the large number of jobs and resources to handle becomes rapidly overwhelming for human operators, and ii) the real performances of the platform that should be taken into account to make robust scheduling decisions. Users usually perform the resource provisioning process manually, i.e they decide all by themselves when and which resources should be claimed and released. This might lead to naive and suboptimal decisions, especially when the number of jobs increases and several resources are proposed at different prices (and with different billing contracts) and performance capabilities. Moreover, the decision parameters, most often the monetary cost and the completion time of the execution, are contradictory. Meeting short completion time constraints implies to provision powerful and numerous resources, and thus pay a high price. On the contrary, a low budget implies more tolerance regarding the completion time.

Though provisioning decisions are difficult to make for human users, IaaS clouds have in common essential characteristics that open interesting opportunities to develop automatic tools to help decision making. Compared to previous large-scale distributed infrastructures (such as grids), IaaS essentially differs by i) its pay-per-use economical model, and ii) the homogeneous performances it contracts with the client. First, the economical model of the cloud is to bill the users solely for the time they have used the resources, based on an atomic time unit that we call Billing Time Unit (BTU) — most often one hour. Second, providers run large clusters where many resources are homogeneous. Even though the hardware differ (for example from one datacenter to another), the provider tends to announce a range of performance rather than a precise hardware type. For example, Amazon has its own power units (called EC2CU) to distinguish the CPU power they reserve to the client's VM depending on the price paid.

In this article, we propose an automatic tool as a project named *Schlouder*[1]. Schlouder is a broker of IaaS resources able to provision and schedule online (i.e, dynamically as jobs arrive, without knowledge of future submissions) independent jobs or static workflows. The provisioning/scheduling is computed after the strategy the user can pick from the existing library. Strategies can be seen as a steering wheel to drive the provisioning process towards performance or cost saving. Our work is not the unique proposal for a client-side broker based on strategies: other works such as [3] have investigated this field and propose other online scheduling strategies which partially overlap our objectives. However, offering a choice of strategies that favor one objective or another may not fully satisfy the user since he has no idea of the alternative solutions. For example, if a user chooses a monetary cost-effective strategy to execute his workload, he will not know how the price paid compares with another plan which would have led to a quicker execution. The system can not give this information because of the online assumption, which only allows to compute alternative schedules *post mortem*. Even *post mortem*, i.e once the workload is known, it is difficult in practice to precisely compute the alternative schedules due to ceiling effects induced by BTUs [4]. To tackle this lack of information, Schlouder brings in a unique feature which is to predict through simulation the makespan and cost of executions under various strategies. The simulation relies on SimGrid [5], a well-established and evaluated discrete event simulator, for which we have written an IaaS-orientated programming interface. Schlouder is, to the best of our knowledge, the only broker with an integrated simulator. This article also contributes to better understand how accurate the predictions produced by simulation can be by comparing the results to real applications runs.

In the rest of the paper, we show in details the contributions made by Schlouder. Section 2 describes the design of Schlouder with its set of provisioning strategies and its simulation module. Section 3 describes the use-cases and experimental setup that serve as the basis for the evaluation, of a real execution (Section 4) first, and then for the simulation of the same cases (Section 5). We analyze in details the behavior of the broker and the accuracy of the prediction computed by the simulation module. Our work is then compared to related work in Section 6 and we sketch possible future work in conclusion.

2. Schlouder: A Client-side Broker, Overview

The motivation behind Schlouder is to assist users in the task of provisioning and scheduling resources for their applications. In a nutshell, Schlouder is a broker able to tailor a virtual platform on-demand, in order to execute a set of jobs according to user's preferences.

[1] Available online at: `http://schlouder.gforge.inria.fr/`

2.1. Schlouder's Architecture

Schlouder orchestrates the cooperation between a *batch scheduler* and a *cloud kit* software to fill our objectives. The role of a batch scheduler (such as OpenPBS or Slurm) is to assign jobs to a set of known resources while an IaaS cloud kit software (such as OpenNebula, OpenStack or Eucalyptus) aims at provisioning the cloud resources by starting or stopping VMs on the physical hosts.

After a job has been submitted, Schlouder's provisioning determines how many resources are needed and it instructs the cloud kit software to start or stop VMs accordingly. Those VMs include the slave part of the batch scheduler software. Once a VM is started, the slave connects to its pre-configured batch scheduling server, which becomes aware of the apparition of a new resource. Schlouder is then able to instruct the batch scheduler to assign a job to its known resources, according to its own scheduling strategies (we actually bypass the scheduling policies of the batch scheduler). Although the batch scheduler is used for only a part of its functionalities, it generally provides a robust framework for handling the job submissions and error logging, which justifies that we did not develop our own specific job submitter.

Schlouder currently supports Slurm [6] as batch scheduler, and OpenStack, Eucalyptus and BonFIRE as cloud kit frameworks.

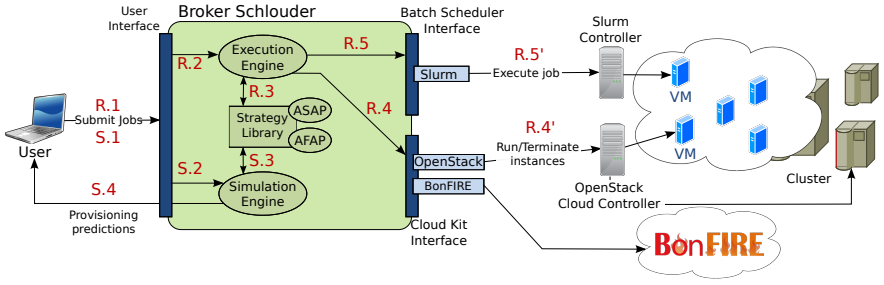

Figure 1: The architecture of Schlouder and its surrounding environment

Figure 1 depicts Schlouder's architecture. A user submit his jobs to Schlouder, represented by the green box labeled 'Broker Schlouder'. Schlouder computes the schedule for each incoming job along the scheduling strategy (described hereafter) chosen by the user. To realize its scheduling decision, the resources are requested to be started or stopped by contacting the computer hosting the cloud kit controller (the figure illustrates a case with two connectors installed, OpenStack and BonFire) using the appropriate API, such as EC2 for Eucalyptus or Nova for OpenStack. The assignment of jobs to resources is then operated through the batch scheduler interface (the Slurm connector on the figure).

2.2. Real Execution or Simulation

Besides this expected function of job execution management, Schlouder brings an original feature allowing the user to predict the result of one schedul-

ing strategy or another through simulation. The core of Schlouder wraps up these two essential modes of working represented on the figure by the *Execution Engine* and *Simulation Engine*. Both engines rely on the provisioning/scheduling algorithms stored in the *Strategy Library*. The respective roles of these core components are:

- The **Strategy Library** (SL) is the placeholder where strategies are stored. Each strategy takes the form of one Perl class that implements the scheduling logics of the strategy. The framework offers the strategies an API to access the characteristics and state of both the platform and jobs. The library currently contains 12 strategies designed after our work [7].
- **T**he **Execution Engine** (EE) is the orchestrator of the broker for a real execution. When a job request is admitted, the EE first gets the provisioning and scheduling decisions from the strategy chosen by the user. It then requests the cloud kit software to start or stop some VMs to reflect the provisioning decision, and then instructs the batch scheduler to schedule the job on the given VM. A background task of the EE is also in charge of monitoring the platform state and the job execution.
- The **Simulation Engine** (SE) is a fully integrated module able to execute a simulation given the workload and a description of the cloud resources. The framework of the simulation will be detailed in Section 2.4.

Depending on whether the submission is a real execution or a simulation, the core components are solicited in the order shown on Figure 1. In case of a simulation, the steps prefixed by S are taken: submission of the job description from the client to the Schlouder broker on the frontend (S.1); translation of the request and set up of the simulation (S.2); simulation of the execution using the strategies picked from the SL (S.3); predictions returned back to the users (S.4). In case of a real submission, the steps prefixed by R are taken: submission of the job description exactly like in S.1 (R.1); translation of the request to the execution engine (R.2); computing of the provisioning and scheduling decision using the strategy chosen by the user (R.3); the EE requests the cloud kit to adjust the platform (start or stop VMs) in agreement with the provisioning decision (R.4); the EE requests the batch scheduler to schedule jobs onto VMs in agreement with scheduling decisions (R.5).

2.3. The Provisioning Strategies

At the core of Schlouder are the strategies for scheduling and provisioning resources. The strategies are heuristics for computing a bi-objective optimization problem, based on monetary cost and makespan of the whole workload. In the pay-per-hour billing model mentioned in introduction (following the *on-demand* Amazon's pricing model), any started VM may leave idle periods until the end of the BTU (hour) that can be re-used for computation for no extra expense, with the counterpart to wait for a VM to become idle. The idea behind minimizing the cost is therefore to compute a schedule that fills in these idle periods.

Hence, a strategy usually implements a heuristic driven towards minimizing or mitigating one or the other objective.

In [7] we designed and studied a dozen provisioning strategies, from the optimally cheaper strategy to the optimally quickest strategy. The algorithms used in these strategies share a common structure composed of two phases:

1. a *deploy* phase, invoked at each job submission. It consists in deciding (1) whether or not a new VM must be deployed, and (2) which active VM the job must be mapped to. It is described in Algorithm 1.
2. a *release* phase, triggered at a parameterized frequency. This release procedure is common to all strategies. It consists in deciding which active VMs must be shutdown and released. Each running VM is examined in turn, and an idle VM is kept running as long as it does not increase the cost. A shutdown occurs when it would incur additional charges.

Algorithm 1 Deploy(j,t)

```
// a new job j is submitted, at date t
C ← ∅ // C is the set of candidate VMs (C ⊂ V)
for v ∈ V do
   if eligible(v, j) then
      C ← C ∪ {v}
   end if
end for
if C ≠ ∅ then
   v ← optimum(C)
else
   v ← deploy() // Create and run a new VM
   V ← V ∪ {v}
end if
enqueue(q_v, j) // Map the job to the VM
```

Where:

- $eligible(v, j)$ is true if j can be assigned to q_v,
- $optimum(C)$ returns the virtual machine to which a job j is to be assigned,
- $deploy()$ provisions and starts a new VM and returns its identifier,
- $enqueue(q_v, j)$ adds the job to the queue of a given VM v. If v is available (i.e q_v is empty) the job actually starts immediately on v without being queued.

The functions *eligible* and *optimum* allow us to define all our provisioning strategies. *eligible* filters out the set of active VMs to which a job can be assigned depending on the current state of VMs. If this set is empty, then a new VM is deployed, otherwise *optimum* selects the VM to assign the job to among the set of candidate VMs. These definitions are summarized in Table 1 and grouped into four families.

- ***1VM4All***: The first strategy provisions a single VM and put all the jobs in its queue. It gives a lower bound on cost for the given workload, as idle time is reused at the maximum.
- ***1VMperJob*-based strategies**: On the opposite side of the spectrum, we devise three "expensive" strategies. *1VMperJob* is a reference strategy for the lowest waiting time possible: it deploys a new VM for each new job request whatever the state of the other active VMs.

strategy	*eligible*(v, j) returns *true*	*optimum*(C) returns $v \in C$ such that ...	comment
1VM4All	always	$v = v_0$	Slowest/Cheapest
1VMperJob	never	any	
1VMperJobPlus		any	Fastest/Most expensive
1VMperJobBest	if $q_v = \emptyset$	s_v is maximum	
1VMperJobWorst		s_v is minimum	
FirstFit		any	Regular BinPacking strategies
BestFit		$i_v - s_v$ is maximum	
WorstFit	if $c(s_v - b_v) = c(s_v - b_v + r_j)$	$i_v - s_v$ is minimum	
EarliestFit		i_v is minimum	+ wait time optimization
RelaxFirstFit$x$		any	Price optimization
RelaxEarliestFit$x$	if $c(s_v - b_v) = c(s_v - b_v + r_j)$	i_v is minimum	+ max wait time constraint
RelaxLastestFit$x$	and $(i_v - t) < (x \times r_t)$	i_v is maximum	

Table 1: The scheduling strategies with their respective parameters for algorithm 1.

- ***Bin-Packing*-based strategies**: We have implemented three classic heuristics for the online bin-packing problem [8], namely *FirstFit*, *BestFit* and *WorstFit*. In our context, *FirstFit* scans the list of already deployed VMs and maps the job to the first VM that does not require to prolongate the rent time over a new BTU, i.e we map the job for no extra-cost. If no such already started VM exits, a new VM is deployed.

  The above strategies have the objective to minimize the number of BTUs. Hence, it tends to minimize the global cost with little consideration to the completion time, which is absent from the original Bin-Packing problem. *EarliestFit* is a first approach to include this criteria. It is a *Fit* heuristic selecting the VM which minimizes the waiting time of the job.

- ***Relax*-based strategies**: *RelaxFirstFit*x, *RelaxEarliestFit*x and *RelaxLastestFit*x include a bound on the waiting time, which is expressed as a factor x. A new VM is deployed when no active VM can handle the job at constant cost or when the waiting time exceeds x times the runtime of the job. A low value of x leads to a *1VMperJobPlus*-like behavior. On the contrary, a high value of x leads to a Bin-Packing-like behavior, as the same delay is considered acceptable.

In this paper, we will restrict our study to two representative strategies from two different families. The first one follows the algorithm of *1VMperJobPlus*, and we will give it hereafter the more mnemonic name *As soon as possible (ASAP)* as it theoretically produces the shortest makespan. Figure 2 illustrates these strategies. The second is *BestFit*, called in the following *As full as possible (AFAP)* which mitigates the cost. In addition, *ASAP* and *AFAP* refine the original *1VMperJobPlus* and *BestFit* strategies by including the boot time parameter. This overhead whose value typically ranges between 30s and 300s [9] has indeed an important impact in real operations. Let us summarize the main line of the two strategies:

- *ASAP* deploys a new VM for each job that can not be handled immediately by an already provisioned VM. We do not deploy a new VM if the boot time exceeds the time to wait for one VM to become available. Thus, the only waiting time is the boot time of the newly deployed VM. On

the example, when the job J_2 is submitted, the VM_1 is idle. The job is immediately executed on VM_1 without any waiting time. However, J_3 and J_4 cannot be executed immediately on VM_1. Hence one new VM is instantiated for these jobs. This execution yields a cost of three BTUs.

- *AFAP* scans the list of already deployed VMs and maps the job to the VM that reduces most its idle time by executing the job. If no such already started VM exists, a new VM is deployed. On the example, a new VM is deployed to map the job J_3 because executing it on VM_1 would trigger a new BTU and then increase the idle time. On the contrary, the execution of J_4 is delayed in order to reduce the idle time of VM_2. This execution yields a cost of two BTUs.

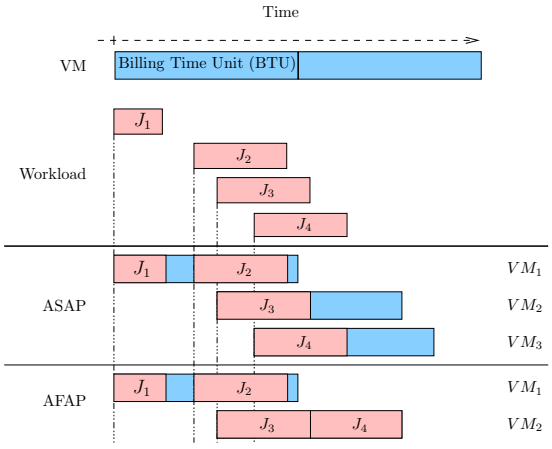

Figure 2: Illustration of the *ASAP* and *AFAP* provisioning strategies on a basic workload composed of four jobs

2.4. The Simulation Framework

The simulation performed by the simulation engine relies on the SimGrid toolkit [5, 10], a well-established discrete event simulator. The inputs to SimGrid are a description of the physical infrastructure (hosts and capabilities of the CPUs, network characteristics and architecture, ...) and the tasks to simulate, expressed either as an amount of floating operations to perform when it is a computation task, or as an amount of bytes when it is a communication (over the network or a storage I/O).

In a distinct project[2], we have added to SimGrid a new interface (an API) that allows its users to describe the operations of an IaaS cloud, such as managing an instance (start, stop, suspend and resume), describing the available

[2] `http://schiaas.gforge.inria.fr`

resources, managing instance types and images, or handling the storage of the IaaS provider. These operations then translate into instructions at the hypervisor level in SimGrid, and then into tasks simulated by the discrete event simulation engine.

In Schlouder, the simulation is performed by an invocation of SimGrid through this interface, and requires as input the platform description and a workload as in a real execution. The workload consists in the jobs with their characteristics (i.e. duration, input and output size, and dependencies of each task). The main output of the simulator is the cost and makespan for each available provisioning strategy. It can also provide very precise information about the chronology of events and the job to VM mapping, which is useful to analyze and improve provisioning strategies post-mortem.

The simulation accuracy relies on three factors: a) the accuracy of the models used by SimGrid for computations and communications, b) the accuracy of the infrastructure description, and c) the accuracy of the tasks' durations. Item a) has been studied in-depth regarding the network model [11], while the CPU model does not yet capture all the complexity induced for instance by the cache effects. Concerning item b), while we can easily capture this information in our private cloud testbeds, it is usually unknown for public clouds. To address this issue, Schlouder offers an automatic discovery of a platform's performances and the generation of the corresponding description, by requesting the execution of an additional monitoring job at submission time. This job executes LINPACK to benchmark the vCPU of the VM, and ping and iperf to benchmark the network between all of the instantiated VMs and between the VMs and the controller. Finally, item c) depends on the information provided by the user. The durations of the tasks may be a source of inaccuracy and we will see in the evaluation section to which extent they change the predictions in our test-cases.

3. Evaluation: Experimental Setup

We now describe the experimental environment setup to evaluate Schlouder. We have chosen two different scientific applications, ran on several different cloud platforms. In each run, we evaluate the two strategies to evidence that they yield contrasted provisioning/scheduling scenarios. We executed each run several times in order to ensure stability between the executions, for a total of 219 runs.

3.1. The Applicative Use-Cases

The two test-case applications are the Open Mass Spectrometry Search Algorithm (OMSSA), in the field of computational biology, and Montage in the field of astronomy. The former is a CPU intensive Bag-of-Tasks (BoT) while the latter is a data intensive workflow. The tasks' durations that are given in each job header are estimations by the user, who is supposedly a practitioner able to evaluate his/her tasks' durations. In our experimental context, we can

easily substitute computed durations for the users' estimations. Indeed, in these two use-cases, the durations are computed using a linear extrapolation based on the data size to compute or communicate. In all applications, we have indeed observed a strong correlation between data size and duration, and a linear regression serves as the basis for this estimation. We have chosen to calibrate the jobs' durations once on our private cloud platform (icps-cloud) and use the same estimations on all platforms despite a slight difference in CPU power, with the objective to test the robustness of the scheduling given this imprecision.

3.1.1. OMSSA

The tandem mass spectrometry analysis (also known as MS/MS analysis) consists in the fragmentation of peptides generated by enzymatic digestion of complex mixtures of proteins. Thousands of peptide fragmentation spectra are acquired and further interpreted to identify the proteins present in the biological sample. Peak lists of all measured peptide masses and their corresponding fragment ions are submitted to database search algorithms such as OMSSA [12] for their identification. Each spectrum is then searched against a database of proteins. We will execute three different kinds of search, representative of common search requests for proteomics data interpretation. It contains up to 257 762 MS/MS spectra interpreted with the following setups: high-resolution measurements and full trypsin cleavage (*hrt*), high-resolution measurements and semi-trypsin cleavage (*hrs*), and finally low-resolution measurements and full trypsin cleavage (*lrt*).

As each search is independent from the others, a natural parallelization consists in making this application a BoT. For each spectra file, a job running OMSSA is created to perform the identification search. All jobs can be run independently on different CPUs. The parallelization grain (i.e., the number of spectra distributed to each CPU) per OMSSA execution is computed as a function of the requested resolution and the number of available CPUs. One "good" granularity has been determined empirically by the proteomists as 10 000 spectra per task for the full trypsin cleavage and 1 250 spectra per task for the semi-trypsin cleavage. The number of tasks vary from 33 for the *lrt* search to 65 for the *hrs* search with a runtime from 1.6 seconds to 485 seconds.

3.1.2. Montage

The second application is the Montage Astronomical Image Mosaic Engine [13], whose objective is to gather astronomical images in Flexible Image Transport System (FITS) format into a mosaic. The input arguments are the desired region of the sky, the size of the mosaic in terms of square degrees, and other arguments such as the FITS image archive to use. In our experiments we used the Two Micron All Sky Survey (2MASS) [14] archive. This application is a workflow composed of three steps: the input images are first reprojected to the coordinate space of the desired output, then any discrepancies in brightness are removed, and finally the different input images are coadded to create the output images.

Figure 3 shows an example of a Montage workflow. Each circle is a task and each arc represents the data dependencies between two tasks. The number in each task represents the level of the task in the workflow. All the tasks at a specific level match the call to a specific Montage command mentioned on the left side of the figure.

In our experiments we chose to compute a mosaic of a unique astronomical object but with different sizes. The chosen object is *Pleiades*, also used in the Montage tutorial. Thereafter, we name NxN with N an integer from 1 to 3, the execution of Montage with 3 sizes.

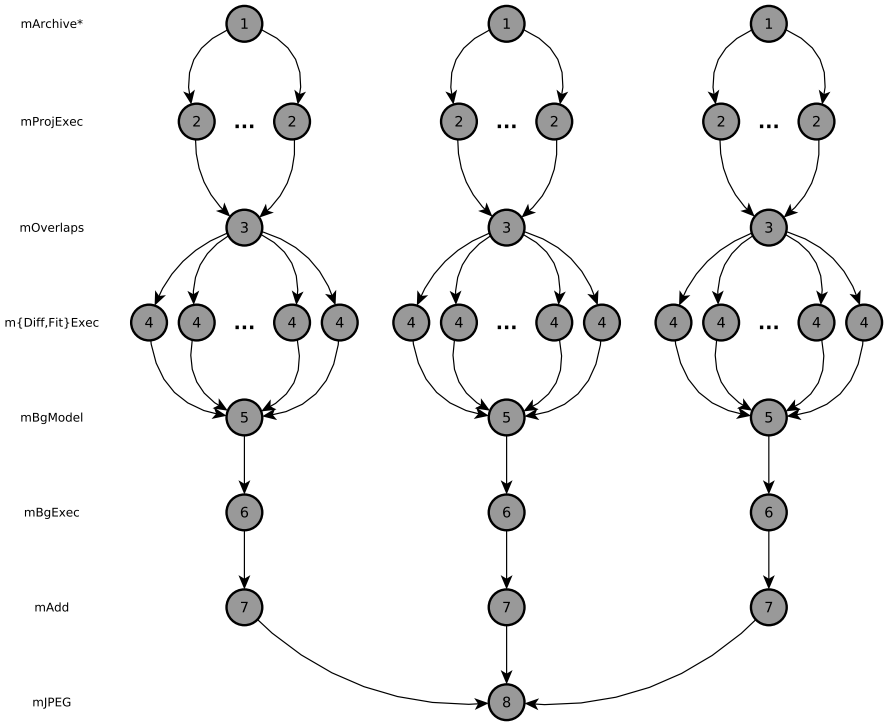

Figure 3: Illustration of a Montage workflow. Each circle is a task and each arc represents the data dependencies between two tasks

We summarize in Table 2 some of the applications' characteristics. The column "# dependencies" is the number of dependencies between the different tasks of the Montage workflow. We evidence that these test-cases are at two ends of the spectrum regarding the resource usage. OMSSA is a compute intensive bag-of-tasks application while Montage is a data intensive workflow.

3.2. The Cloud Platforms

We executed the applications on two different clouds: a private cloud in our lab based on OpenStack, and a public cloud, namely BonFIRE [15], targeting

Table 2: OMSSA and Montage characteristics

		# tasks	walltime (s) min	avg	max	data size	communication to computation ratio	# dependencies
OMSSA	*lrt*	33	86	152	398	1.2GB	2%	0
	hrt	34	6	16	199	514MB	16%	0
	hrs	65	66	148	411	1.3GB	1%	0
Montage	1x1	163	4	18	167	8GB	94%	288
	2x2	467	6	49	593	30.9GB	97%	863
	3x3	993	4	60	1 340	69.8GB	98%	1 861

large scale cloud research.

3.2.1. Public Cloud

BonFIRE [15] is a public multi-cloud testbed, currently operated on seven geographically distributed sites across Europe. We ran our experiments on three of them: de-hlrs, fr-inria, and uk-epcc. Each site is accessible through a common API based on OCCI [16]. The cloud kit is OpenNebula 3.6 in a version derived for BonFIRE. These testbeds have different physical infrastructures[3]. The maximum number of VMs we can instantiate depends on the current site usage, the disk quota, and the CPU quota. In our case, using a 10GB VM image with 1 core and 1024 MB memory, we were able to run from 20 to 23 VMs on each site. Notice that this testbed is not smaller than what most public clouds offer since they generally impose a limit on the number of instances per user (for instance 20 at EC2, although this limit can be lifted through a request form). The central cloud storage provided by BonFIRE is a Network File System (NFS) share. The server is located on the be-ibbt testbed. In order to access the data, a resource need to be in the BonFIRE WAN and to mount this shared storage. The Schlouder server is in the BonFIRE WAN through a VPN. Thereafter, we name these sites BonFIRE-*site_name*.

3.2.2. Private Cloud

Our private cloud setup is composed of two local servers, each being a dual 2.67GHz Intel Xeon X5650 processor, with 12 hyper-threaded cores on both CPUs. We run up to 25 single core VMs with 1024 MB memory. Both servers run on a Linux 3.0.0-12 with the KVM module. For this experiment, we used OpenStack 2012.1.3-dev [17] as cloud kit. Our central cloud storage is the S3-compliant software Walrus installed on a different server, on the same LAN. Thereafter, we name this cloud icps-cloud.

Table 3 shows the characteristics of the private and the public cloud. They differ in many regards: they are of different size and CPU power, and they use different hypervisors. On BonFIRE, the physical machines are heterogeneous

[3]Comprehensive information is available online`http://www.bonfire-project.eu/infrastructure/testbeds`

Table 3: Characteristics of our four testbeds

cloud	# nodes	# cores	CPU power (GHz)			# max VM	hypervisor	storage	boot time (s)		
			min	avg	max				min	avg	max
icps-cloud	2	48	2.67	2.67	2.67	25	KVM	Walrus	36	120	255
BonFIRE-de-hlrs	36	344	1.65	2.15	3.40	20	Xen 3.1.2	NFS	34	266	2 097
BonFIRE-fr-inria	4	96	0.93	0.93	0.93	20	Xen 3.2	NFS	123	982	11 084
BonFIRE-uk-epcc	7	176	1.29	1.42	1.54	20	Xen 3.0.3	NFS	130	437	1 047

Table 4: Makespans (in s) and costs (in BTU) for each strategy (average over all runs, all platforms)

		hrs	*hrt*	*lrt*	1x1	2x2	3x3
Makespan	*ASAP*	1 127	378.8	826.1	1 976	7 254	15 484
	AFAP	4 707	612.8	4 132	3 229	9 916	17 415
	ratio	4.18	1.62	5	1.63	1.37	1.12
Cost	*ASAP*	21.7	8.5	20.7	8.91	31.3	52.9
	AFAP	5.19	1	2.55	1.36	8.09	16.9
	ratio	0.24	0.11	0.12	0.15	0.26	0.32

with very different CPU power. The private cloud has a constant and low boot time while the public cloud showed occasional irregular very long boot times.

4. Evaluation of Real Executions

We now analyze the behavior of Schlouder in the experimental conditions defined in the previous section. We verify whether the two strategies fulfill their respective objectives when ran for applications on actual infrastructures. We first report the general behavior under a strategy, by reporting the average makespans and costs for the complete executions of the bags-of-tasks (OMSSA) and workflows (Montage) in Table 4.

We see that *AFAP* does spare a significant part of the cost of *ASAP* for the two types of applications. Conversely, the execution time is longer, though this increase varies largely depending on the application type. For the computation intensive workloads, the increase may reach 3 to 4 times the makespan of *ASAP* while the workflow, which is communication intensive, only takes from 12% to 63% more time to complete. OMSSA *hrt* which also exhibits more communications than *hrs* and *lrt*, has a makespan overhead no bigger than the workflow executions. Therefore, it appears that *AFAP* is overall a competitive strategy with communication intensive applications.

4.1. Job-level Analysis on Different Platforms

We now compare the average behavior of jobs on each platform in order to evidence the impact of different infrastructure characteristics on the strategies. We define the following metrics to characterize a job's behavior. A job executes for a *walltime*, composed of:

- The *wait time*, which consists in the time taken to boot VMs (*boot wait time*) and the time jobs wait before they can actually run after they have been scheduled by the batch scheduler (*schedule wait time*). The boot time is possibly null if the VM was already running. The schedule wait time includes the time a job has to wait because of a dependency to another job in case of a workflow. It also includes the time to wait for a VM to become available in case the maximum number of VMs are already in use.
- The *communication time* represents the durations of the data transfers required by inputs and outputs of jobs.
- The *execution time* is the duration of the computation.

The overhead of Schlouder's management tasks has been measured but is negligible (on average 0.42% of the execution time) and not shown on the figure.

Figure 4 shows the walltime time breakdown for the execution of the six applications[4]. Times can be read on the left vertical axis. Also presented are the costs, in numbers of BTU consumed, to be read on the right vertical axis.

4.2. Observations

The first observation is that the strategies rank in the same order whatever the platform, excepted for the case BonFIRE-fr-inria using *ASAP*, which will be discussed below. *AFAP* (plots in the right column on Figure 4) reduces the cost in all cases: it saves from 64% to 94% as compared to *ASAP*. For instance, the average number of BTUs over all platforms for the 2x2 execution is respectively 8 and 31 BTUs. In all cases but the exception BonFIRE-fr-inria, *AFAP* yields as expected a longer makespan, due to the longest schedule wait times. The two computation intensive applications *hrs* and *lrt* exhibit the biggest difference in schedule wait time when comparing *ASAP* and *AFAP*.

The second observation is that for a given strategy, the breakdown of the time spent is proportionally similar from one platform to another. For instance the ratio of the schedule wait time of *AFAP* to *ASAP* for 1x1 varies between 3.11 (fr-inria) and 3.63 (de-lrs), or for *lrt* between 9.02 (de-lrs) and 10.79 (uk-ecc).

Thirdly, this job-level analysis also reveals that *ASAP* incurs a longer boot time on average. The reason is that many VMs are booted simultaneously, inducing network and/or bus contention to load the VM images.

Last, we notice a relatively small part spent on average in communication time, even for communication intensive applications. This is explained by the structure of the Montage workflows, that spawn a large number of tasks (see Table 2), of which only the final ones gather a large amount of data through communications, hence the small average of communication time. We can also notice that the average communication time is constantly larger with *ASAP*, probably because communications have a greater overlap inducing contention,

[4]Note that due to maintenance operations on BonFIRE-fr-inria at the time of the experiments, we were not able to execute the 3x3 experiment using the *ASAP* strategy.

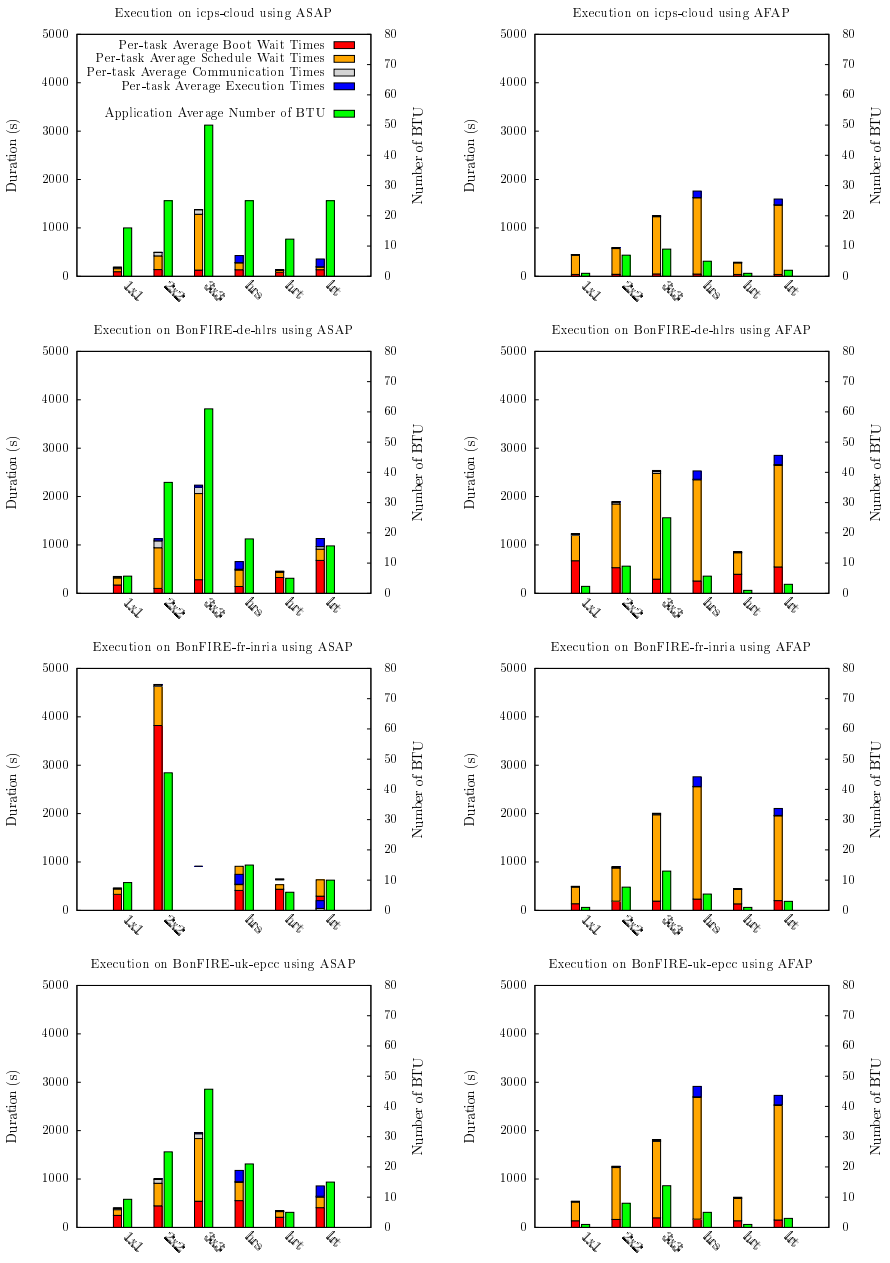

Figure 4: Breakdown of the walltime of the execution of OMSSA and Montage on different platforms, using the *AFAP* and *ASAP* provisioning strategies

whereas *AFAP* spares almost all of this contention. This is the reason why *ASAP* fails at completing in a shortest makespan in some situations, as we will see in the next section.

Finally in this experiment, the wide range of platforms and the different types of applications do not involve divergent behaviors from the broker. This is the encouraging sign that Schlouder produces expected results with little or even no calibration of the target platform.

4.3. Pathological Cases

While the overall observation is that Schlouder fulfills the objectives targeted by the two strategies, the experiment exhibit unexpected behaviors in some specific situations. We investigate hereafter the causes of these peculiarities.

Boot Time Variation. The 2x2 execution with *ASAP* (Figure 4) results in considerably longer wait times on BonFIRE-fr-inria (averagely 9 400 s per task) as compared to the other platforms (from 1 095 s to 2 660 s). Nearly half of this wait time is actually due to boot times. This difference is explained by the way BonFIRE-fr-inria provisions new VMs in response to a user's request. By comparing BonFIRE-fr-inria and BonFIRE-de-hlrs provisioning to the same request, we discovered that both clouds eventually provided 22 VMs, but while these VMs were started simultaneously on BonFIRE-de-hlrs, they were started in three consecutive batches of 8 simultaneous VMs maximum at BonFIRE-fr-inria. The timelines of the executions showing the number of concurrent jobs or VMS running (vertical axis labeled *diameter*) at the two sites of 2x2 using *ASAP*, presented in Figure 5, evidence this behavior. This situation actually happens when the infrastructure is over capacity. Most of the infrastructures we know return an error in such a case. On BonFIRE, the request remains pending, waiting for the physical infrastructure to be able to deploy the VM. We keep this case to highlight the issue, but Schlouder has actually been patched with a timer to circumvent such a case.

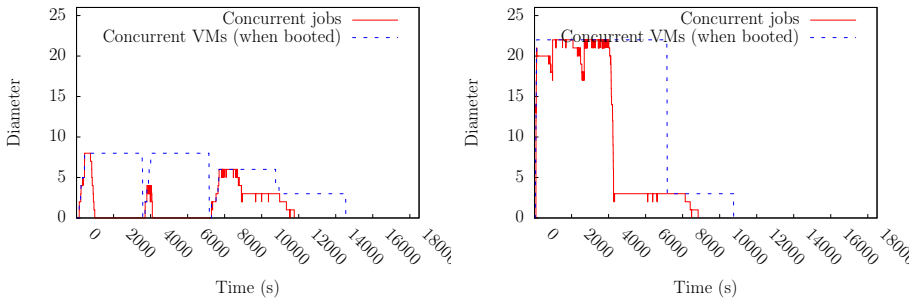

Figure 5: 2x2 executions using *ASAP* on BonFIRE-fr-inria (left) and BonFIRE-de-hlrs (right)

ASAP may fail to speed-up execution. *ASAP* does not result in a shorter makespan everytime. This might surprise the user given the objective of the strategy. For example, for montage 3x3, which is a communication intensive application (see Table 2) its execution using *ASAP* or *AFAP* lasts approximately the same time, about 6 000 s on icps-cloud. By plotting the execution timeline showing the number of concurrent VMs on Figure 6, we see that *ASAP* provisions a higher number of VMs than *AFAP* (25 against 6). *ASAP* increases the parallelism of tasks and thereby increases the concurrency between the communications. In this case – this phenomenon is not observed for 1x1 and 2x2 —, the contention over the network results in a slowdown of the workload execution walltime. Thanks to the Simulation Engine, users can avoid such bad surprises.

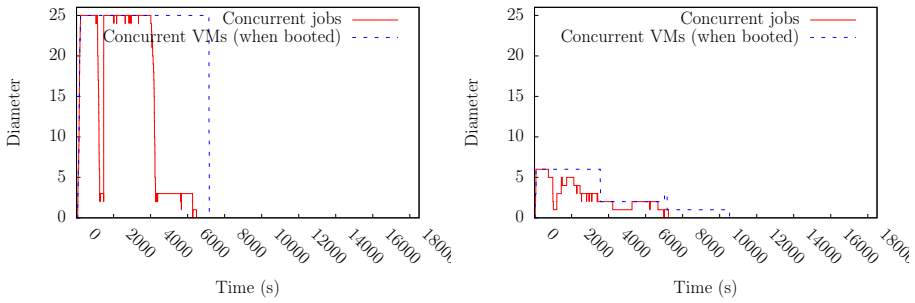

Figure 6: 3x3 executions on icps-cloud with *ASAP* (left) and *AFAP* (right)

Tasks Walltime Mis-estimate. Strategies compute their schedule based on the job walltime, which will call estimation, told by the user in the preamble of the job description. Recall (see section 3.1) that these data were obtained by calibration of the jobs on the icps-cloud platform. We observe that the coarse granularity of BTUs offers enough tolerance to leave the scheduling unchanged in most cases. However, we could exhibit in the experiments the situation represented in Figure 7. The execution of *lrt* on icps-cloud, where the computation was calibrated, required two VMs as estimated by the *AFAP* strategy. When running it on BonFIRE-de-hlrs, where the CPU power is slightly lower, the strategy made its decision based on an under-estimated walltime (4 810 s instead of 3 435 s), which led to overload the BTU capacity (3600 s) and to trigger an extra BTU. Accurate walltime estimations would have made *AFAP* to provision an extra VM instead, reducing the completion time while keeping the same number of BTUs.

The walltime estimation is a classical issue in the field of batch scheduling, and some scheduling algorithms are more sensitive than others to tasks duration uncertainty [18]. For *AFAP* — and more generally for strategies based on bin-packing — users should provide accurate durations to ensure an effective execution of the strategy. Otherwise, users could turn to less sensitive algorithms such as *ASAP*. However, many techniques can be used to improve estimation accuracy. For example, in our case, we can either make some calibrations on

every platform, learn from past executions, or infer the estimations according to the platform performances as reported by Schlouder's monitoring jobs.

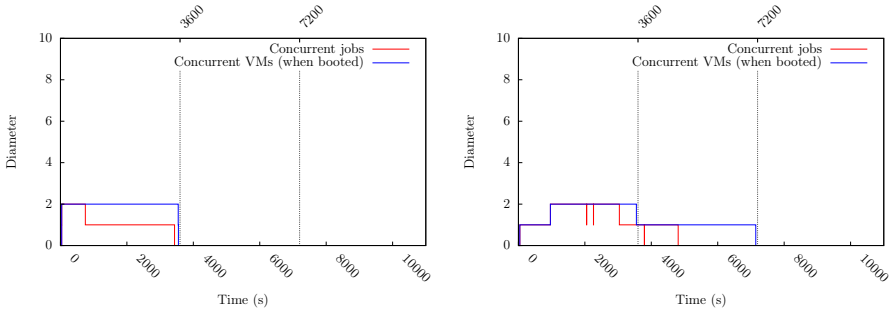

Figure 7: *lrt* execution using *AFAP* on icps-cloud (left) and on BonFIRE-de-hlrs (right)

5. Evaluation of the Simulation

5.1. Raw Accuracy

Our objective is to understand if the predictions produced through simulation are accurate enough for the user to make relevant decisions. In our experiments, we gathered 219 results of real executions (see Section 3) to which we compare the predictions. We define the divergence between the observed time and cost and those predicted by simulation as $err = \frac{|m-m'|}{m}$, where m is a real observation and m' is obtained by simulation.

We have discussed in Section 2.4 the sources of inaccuracy for the simulation. Two of the factors should be examined in our experiment to understand possible divergences. The first factor concerns the characteristics of the platform. In our experiment, we choose to use the information gathered by the Schlouder monitoring job to automatically build the SimGrid platform file. The motivation for this choice is that it could also be used on a public cloud and that it reflects the actual state of the platform. The second factor is the tasks' durations input to the simulator. These durations are generally estimated by the user and may therefore represent an important source of error. In the analysis hereafter, we isolate this factor by observing the simulation results when

- the tasks durations are estimated by the user, as explained in section 3.1, and we call this scenario simulation with *estimated durations*,
- the tasks durations input to the simulation are those observed in real execution, and we call this scenario simulation with *consolidated durations*.

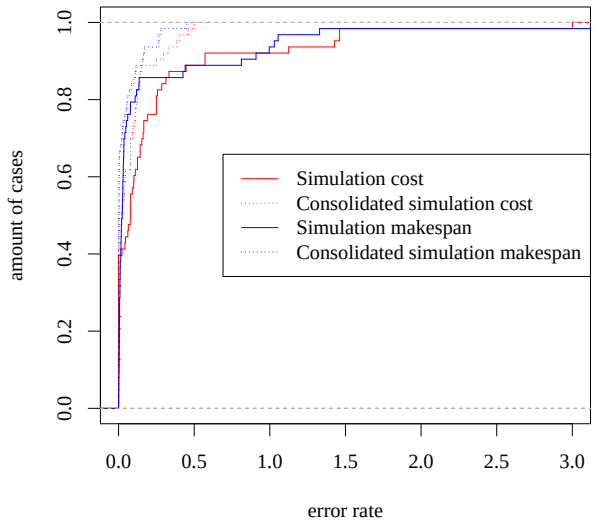

Figure 8: Simulation Engine accuracy in terms of cost and makespan for estimated and consolidated task durations

Simulation with Estimated Durations. The continuous line in Figure 8 shows the cumulative distribution function of the error rate in term of cost, and makespan, that is the walltime. The cost is perfectly predicted for half of the 219 cases, and the error rate is kept under 0.5 for almost 95% of them. The makespan error rate is kept under 0.5 for 87% of the jobs. Both reach a maximum error rate of 3.0.

Simulation with Consolidated Durations. If we now replace the estimated durations of tasks with observed ones, the mispredictions drastically fall down, as shown by the dashed line in Figure 8. The error rates in term of cost and makespan are kept under 0.1 for almost 96% of the cases. The difference with the estimated durations includes both the deviation in the computation time but also some network data transfers over NFS, which are difficult to capture on the BonFIRE platforms. The storage for this testbed is actually shared and located in Belgium, which implies cross-traffic and long network communications. This architecture is especially difficult to model, and much more complex than most public clouds which generally provide storage local to the datacenter. The communication prediction remains therefore an issue to investigate. However, the simulation process itself appears to be valid if accurate task durations are provided by the user.

5.2. Accuracy of Strategy Ranking

Beyond these raw measures of divergence between simulation and real execution, we believe simulation is most useful when it allows the user to compare a portfolio of strategies. Most important in this case will be the ranking of strategies regarding each objective. Small mispredictions will not necessarily make the user change his mind as long as the results preserve the ranking between strategies. We now analyze the accuracy of simulation in this regard by defining a new metric based on ratios on the objectives.

We define the efficiency e of $ASAP$ compared to $AFAP$, as the ratio of makespan gain to cost gain.

$$costup = cost_{ASAP}/cost_{AFAP} \quad (1)$$

$$speedup = Makespan_{AFAP}/Makespan_{ASAP} \quad (2)$$

$$e = \frac{speedup}{costup} \quad (3)$$

Hence the following example values of e means:

- $e = 1$: $ASAP$ increases the cost as much as it speeds up the execution, e.g. The user pays twice more using $ASAP$ to halve the completion time.
- $e > 1$: $ASAP$ increases the cost less than it speeds up the execution, e.g. The user pays 1.5 more using $ASAP$ in order to halve the completion time.
- $e < 1$: $ASAP$ increases the cost more than it speeds up the execution, e.g. The user pays 3 more using $ASAP$ in order to halve the completion time.

Efficiency of Simulation with Estimated Durations. As for the raw measures comparison, we first evaluate the relevance of simulation by comparing the efficiency of real executions versus the one predicted by simulation with estimated durations. Drawn as a continuous line on Figure 9 is the cumulative distribution function of the error rate of the $ASAP$ efficiency between the real executions and simulations. It is perfectly predicted for half of the 219 cases, and the error rate is kept under 1.0 for 81% of them. However, a small fraction of the simulations show a considerable deviation to real efficiency, with a maximum of 7.2, and needs further investigation.

Efficiency of simulation with Consolidated Estimations. If we now use the real walltimes in the simulation to compare simulation results against the provisioning and scheduling done in the real executions, we obtain the cumulative distribution function plotted as a dashed line on Figure 9. We can see that there is almost no error in the efficiency prediction. Therefore, our conclusion is that, as long as the user can provide accurate estimations for the task's durations, the simulation produces the correct ratio makespan/cost.

During this study, we had to understand some intricacies to explain the reason for remaining inaccuracies. First, the overhead of job management (which

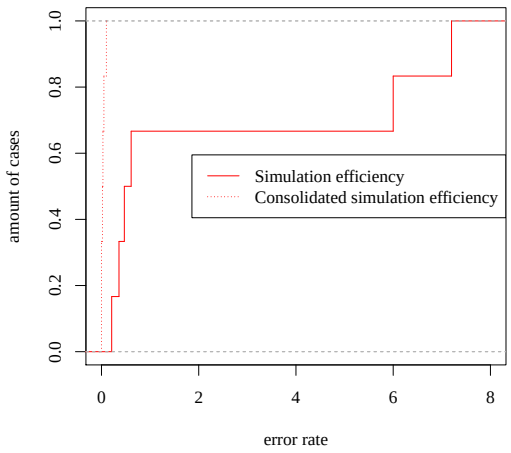

Figure 9: Simulation Engine efficiency accuracy for estimated and consolidated task durations

includes Schlouder, the cloud kit and the batch scheduler) is not simulated, however the small errors on the makespan it implies is not significant (under 1%). Second, these experiments revealed a bug in Schlouder that may cause erroneous provisioning decision in rare situations. This bug is caused by the simultaneous execution of the provisioning and submission thread, making the action of provisioning to possibly happen in the middle of the batch submission process when handling very large batches. For technical reasons, this resulted into boot time mispredictions. The threads interleaving made the bug very difficult to track down. We identified this bug at the light of the simulation analyzed with the BTU viewer included in Schlouder's tools. The bug has been corrected since then and we have further checked that the only significant errors between the simulation with estimated durations and real execution were due to this bug. Finally, we conclude that the Simulation Engine (SE) is very accurate as soon as the walltimes can be accurately estimated. Moreover, the SE proved to be a useful tool to check the implementation correctness of a strategy.

6. Related Work

We can see Schlouder as a service that abstracts the underlying infrastructure, and we may wonder how our approach relates to the usual IaaS-PaaS-SaaS categorization. Schlouder actually lies midway between IaaS and PaaS. It differs from a PaaS regarding the way the infrastructure is abstracted. In PaaS, the infrastructure is handled internally by the provider and clients are totally blind regarding the way their applications will share resources (e.g VMs with other users). An IaaS client-side broker has a finer knowledge and control over the re-

sources it chooses on behalf of its user. Another difference is that PaaS imposes specific development frameworks, which have been tailored by the provider to accommodate its PaaS. There is no such restriction in an IaaS broker since all applications can be freely embedded into a VM image. More generally, operating IaaS clouds allows for a much larger range of usages, including exploiting infrastructures composed of several clouds.

Thus, Schlouder is a hybrid, closer to IaaS than to PaaS. It automates VM scaling, provides limited cloud interoperability, and has a simulation-based recommendation system for choosing the appropriate provisioning strategy. Therefore, in the following related work description, we cover both IaaS solutions and specific PaaS projects whose target applicative field is comparable with the one addressed in this paper.

The problem of executing a workload of BoTs or workflows on the cloud has been extensively studied. In our review of some of these projects, we are particularly interested in their ability to provide clients with customized provisioning and scheduling strategies; whether or not they restrict users to an API or programming model; and their availability to the open source community. We include in our comparison solutions offering brokers on both single and multiple cloud platforms. Interestingly, most brokers operate on an IaaS basis as they require a direct access to the infrastructure. Brokers exist in some PaaS solutions as well but their behavior is usually hidden from the client. Finally while most commercial PaaS systems are restricted to a single provider, some have begun adding intercloud capabilities [19, 20]. We divide the related work in two parts: (1) PaaS with automatic provisioning and (2) client oriented IaaS brokers.

6.1. PaaS solutions with automatic provisioning

Projects like Aneka [19], mOSAIC [20] or COMP Superscalar [21] together with many commercial PaaS (e.g., Google App Engine [22]) have two constraints from the client point of view which do not occur in our proposal.

First, they completely hide access to the virtual resources and provide inbuilt automatic or manual scaling. This impacts users that want to take advantage of PaaS services but maintain some control of the VMs from an optimization point of view. Hybrid PaaS like Azure [23] offer a partial solution but still do not allow clients to customize the scheduling policy.

Second, they restrict users to specific APIs, tools or programming models. For instance, Tejedor et al. [21] propose a platform and a programming model based on Java annotation to execute an application on an IaaS cloud. Vecchiola et al. [19] and Petcu et al. [20] propose a more comprehensive platform with tools to manage the workloads but still confine clients to specific programming models and APIs. Commercial solutions like Google App Engine support mostly web applications and only a few languages.

6.2. Client oriented IaaS brokers

The cloud computing has first been developed for a web usage and is still widely used in this context. Thus a lot of research has been made to build a cloud broker to serve web services. STRATOS [24], Kingfisher [25], CompatibleOne [26] and FCM [27] are examples of frameworks dealing with inter-cloud web based applications. STRATOS is tested on both Amazon EC2 and Rackspace to show its cross-cloud ability. The broker can choose the best resources in the best clouds based on the topology of the application and a set of user specified objectives. Kingfisher supports private and public clouds and uses a cost-aware provisioning strategy that cannot be modified or changed. CompatibleOne is both a model to describe the users' needs (CORDS) and a cloud broker to provision and deploy cloud application (ACCORDS). The latter is of interest for us. It offers an interoperable way to run instances on multiple clouds. The user can customize the provisioning strategy by specifying an agreement document, combination of CORDS and WebService Agreement [28]. Kertesz et al. [27] propose Federated Cloud Management (FCM) a cloud broker for stateless web-services in a multi-cloud environment. Their broker makes the provisioning decision based on multiple criteria: the number of incoming request, monitoring results, SLA constraints and billing period of the IaaS provider. However it does not allow to customize the provisioning strategy.

None of these projects provides a method for handling other types of applications such as scientific jobs. Web requests can be assimilated with short jobs with high reactivity. For this reason brokers supporting them must provide mechanism for quickly adapt to peak demands. In contrast, scientific workloads have no reactivity constraints but their duration is highly dependent on the application type. Although not an IaaS, Google App Engine provides a good example in this direction. It restricts clients from making requests that take longer than 30 seconds.

Brokers for general scientific computations have been investigated in [1], [2], [29], and [3]. The main difference with our solution for the three following works is that they do not provide a method of customizing the provisioning policy. In [1] a cluster architecture to deliver flexible and elastic High Throughput Computing environments is presented. The aim is to grow a local cluster capacity using an external cloud provider. The results demonstrate that only the overheads due to virtualization affect the performance of the elastic cluster. TorqueCloud [2] is a tool built with the distributed resource management software TORQUE and the Eucalyptus cloud platform. A deadline constraint algorithm, IdleCached, for BoT is presented. A difference is that they aim to extend TORQUE while we designed a modular platform allowing to switch to different cloud providers, schedulers and provisioning strategies. Mendez et al. [29] designed a solution to offer to scientists a SaaS platform for scientific computation named e-clouds. Their platform is used by the scientists through a web portal. With this design, e-clouds restrict the user to execute one application chose in a finite list whereas we let the user configure his own VM image with the software he chooses. Moreover, the software does not seem to be publicly available.

Two tools close to our are the Virtual Execution Platform (VEP) designed by Harsh et al. [30] and a tool that aims at testing user-defined strategies designed by Villegas et al. [3]. VEP [30] is part of the Contrail platform and aims to help the cloud provider to become part of the larger Contrail cloud federation. It eases the execution of an application through a web portal. The two main differences are the lack of recommendation system like the SE and the requirement to install a piece of software on the cloud provider side.

The work by Villegas et al. [3] is the closest to ours. Their SkyMark performance evaluation tool is able to generate, submit, and monitor bags-of-tasks to IaaS clouds conforming to the EC2 API. Eight provisioning and four allocation strategies are presented and tested. While they also developed a discrete event simulator that duplicates the SkyMark functionality, the simulator does not have the maturity of the SimGrid tool (SkyMark is not available for testing). Further, the simulation process is not intended to be integrated into the framework as a recommendation utility, as we do with Schlouder.

Among commercial brokering systems, one of the most well-known is RightScale [31] which allows its clients to deploy and monitor VMs across multiple cloud providers *via* a web interface. The clients define the rules which automatically trigger the VM provisioning, based on thresholds for various monitored conditions such as the load. This solution does not provide a way to customize the provisioning strategy based on other criteria than threshold. Other companies such as Amazon Elastic Beanstalk [32], CloudFoundry [33] or Windows Azure [23] provide similar services but without multicloud support.

7. Conclusion

We have presented in this paper our cloud broker Schlouder, which helps users drive their IaaS resources provisioning and scheduling for BoTs or workflows. Its usage has been illustrated through the execution of two real applications in bioinformatics and astronomy.

We highlighted that Schlouder, using one of the embedded provisioning strategies, is able to effectively favor one of the criterion, cost or makespan. We have studied the behavior of *AFAP* strategy to take advantage of the pay-per-use property in order to provision a limited amount of VMs and use all the rented BTUs, but many intermediate strategies are also available in Schlouder.

The simulation of an application execution on a cloud is a challenging issue. However, this task is eased by the homogeneity of the performances of the cloud. To this end, Schlouder offers a simulation engine (SE) able to accurately predict the makespan, cost and the jobs-to-VM mapping of using such or such strategy, and how the strategy will compare to the others. Although the SE requires a description of the real platform, which is often not known in the case of public clouds, the black box nature of the cloud makes possible to automatically discover the available bandwidth, latency and the CPU power of the resources, and from these data, build a model of the real platform.

The future work is threefold. First, we will tackle with the issue of choosing the right provisioning strategy which might be a difficult choice for the user,

even with the help of SE. Our idea is to design one unique meta-strategy, which would apply the most inexpensive strategy among all available ones, according to one user given deadline. Indeed, while cost/makespan tradeoffs are difficult to apprehend, deadlines are common knowledge for scientists. Second, we will improve the SE results by designing a P2P monitoring module, able to share information about both public cloud infrastructures and application runtimes among the Schlouder users. Last, in order to circumvent the great impact of the walltime prediction on the *AFAP* provisioning strategy, we plan to add a measure of confidence to the SE results. Knowing the error rate e of the walltime prediction, we can simulate the execution of the user's workload with both $walltime\_prediction \times (1 + e)$ and $walltime\_prediction \times (1 - e)$. Hence we obtain an interval of the possible results, and thus a measure of confidence of the SE results.

Acknowledgements

This work is partially supported by the ANR project SONGS (11-INFRA-13) and by the French ministry of defence - Direction Générale de l'Armement.

This publication contains results generated using the BonFIRE multi-site cloud facility. The authors wish to thank the BonFIRE Foundation for their help and support.

This research made use of Montage, funded by the National Aeronautics and Space Administration's Earth Science Technology Office, Computation Technologies Project, under Cooperative Agreement Number NCC5-626 between NASA and the California Institute of Technology. Montage is maintained by the NASA/IPAC Infrared Science Archive.

[1] R. S. Montero, R. Moreno-Vozmediano, I. M. Llorente, An elasticity model for high throughput computing clusters, Journal of Parallel and Distributed Computing 71 (6) (2011) 750–757. doi:10.1016/j.jpdc.2010.05.005.

[2] H. Song, J. Li, X. Liu, IdleCached: an idle resource cached dynamic scheduling algorithm in cloud computing, in: 9th UIC and ATC, 2012, pp. 912 –917.

[3] D. Villegas, A. Antoniou, S. Sadjadi, A. Iosup, An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds, in: CCGrid, IEEE, 2012, pp. 612–619.

[4] E. Michon, J. Gossa, S. Genaud, Free elasticity and free cpu power for scientific workloads on iaas clouds, in: 18th International Conference on Parallel and Distributed Systems (ICPADS), IEEE, 2012, pp. 85–92.

[5] H. Casanova, A. Legrand, M. Quinson, Simgrid: a generic framework for large-scale distributed experiments, in: 10th IEEE International Conference on Computer Modeling and Simulation, 2008.

[6] A. B. Yoo, M. A. Jette, M. Grondona, Slurm: Simple linux utility for resource management, in: JSSPP, 2003, pp. 44–60.

[7] S. Genaud, J. Gossa, Cost-wait trade-offs in client-side resource provisioning with elastic clouds, in: 4th International Conference on Cloud Computing (CLOUD), IEEE, 2011.

[8] E. G. Coffman, M. R. Garey, D. S. Johnson, Approximation algorithms for bin packing: a survey, PWS Publishing Co., Boston, MA, USA, 1997, pp. 46–93.
URL http://www2.research.att.com/~dsj/papers/BPchapter.ps

[9] M. Mao, M. Humphrey, A performance study on the vm startup time in the cloud, in: 5th International Conference on Cloud Computing (CLOUD), IEEE, 2012, pp. 423–430.

[10] H. Casanova, A. Giersch, A. Legrand, M. Quinson, F. Suter, Versatile, scalable, and accurate simulation of distributed applications and platforms, Journal of Parallel and Distributed Computing 74 (10) (2014) 2899–2917.
URL http://hal.inria.fr/hal-01017319

[11] P. Velho, L. M. Schnorr, H. Casanova, A. Legrand, On the validity of flow-level tcp network models for grid and cloud simulations, ACM Trans. Model. Comput. Simul. 23 (4) (2013) 23. doi:10.1145/2517448.
URL http://doi.acm.org/10.1145/2517448

[12] L. Y. Geer, S. P. Markey, J. A. Kowalak, L. Wagner, M. X. D. M. Maynard, X. Yang, W. Shi, S. H. Bryant, Open mass spectrometry search algorithm, J Proteome Res. 3 (5) (2004) 958–964.

[13] J. C. Jacob, D. S. Katz, G. B. Berriman, J. C. Good, A. Laity, E. Deelman, C. Kesselman, G. Singh, M.-H. Su, T. Prince, et al., Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking, International Journal of Computational Science and Engineering 4 (2) (2009) 73–87.

[14] IPAC, Two micron all sky survey, (accessed August 13rd 2014).
URL http://www.ipac.caltech.edu/2mass/

[15] K. Kavoussanakis, A. Hume, J. Martrat, C. Ragusa, M. Gienger, K. Campowsky, G. Van Seghbroeck, C. Vázquez, C. Velayos, F. Gittler, et al., Bonfire: the clouds and services testbed, in: 5th IEEE International Conference on Cloud Computing Technology and Science, Cloudcom, 2013.

[16] Open cloud computing interface, (accessed August 13rd 2014).
URL http://occi-wg.org/

[17] Openstack cloud software, (accessed August 13rd 2014).
URL http://www.openstack.org/

[18] L.-C. Canon, E. Jeannot, Evaluation and optimization of the robustness of dag schedules in heterogeneous environments, IEEE Trans. Parallel Distrib. Syst. 21 (4) (2010) 532–546.

[19] C. Vecchiola, X. Chu, R. Buyya, Aneka: a software platform for .net-based cloud computing, High Speed and Large Scale Scientific Computing (2009) 267–295.

[20] D. Petcu, B. Di Martino, S. Venticinque, M. Rak, T. Máhr, G. E. Lopez, F. Brito, R. Cossu, M. Stopar, V. Stankovski, et al., Experiences in building a mosaic of clouds, Journal of Cloud Computing: Advances, Systems and Applications 2 (1) (2013) 12.

[21] E. Tejedor, J. Ejarque, F. Lordan, R. Rafanell, J. Alvarez, D. Lezzi, R. Sirvent, R. M. Badia, A cloud-unaware programming model for easy development of composite services, in: Cloud Computing Technology and Science (CloudCom), IEEE, 2011, pp. 375–382.

[22] Google app engine, (accessed August 13rd 2014).
URL `https://developers.google.com/appengine/`

[23] Windows azure, (accessed July 17th 2013).
URL `http://azure.microsoft.com`

[24] P. Pawluk, B. Simmons, M. Smit, M. Litoiu, S. Mankovski, Introducing stratos: A cloud broker service, in: 5th International Conference on Cloud Computing (CLOUD), 2012, pp. 891 –898. doi:10.1109/CLOUD.2012.24.

[25] U. Sharma, P. Shenoy, S. Sahu, A. Shaikh, A cost-aware elasticity provisioning system for the cloud, in: 31st International Conference on Distributed Computing Systems (ICDCS), 2011, pp. 559 –570.

[26] S. Yangui, I.-J. Marshall, J.-P. Laisne, S. Tata, CompatibleOne: the open source cloud broker, Journal of Grid Computingdoi:10.1007/s10723-013-9285-0.

[27] A. Kertesz, G. Kecskemeti, M. Oriol, P. Kotcauer, S. Acs, M. Rodríguez, O. Mercè, A. C. Marosi, J. Marco, X. Franch, Enhancing federated cloud management with an integrated service monitoring approach, Journal of grid computing 11 (4) (2013) 699–720.

[28] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, M. Xu, Web services agreement specification (ws-agreement), in: Open Grid Forum, Vol. 128, 2007.

[29] D. Mendez, M. Villamiazr, H. Castro, e-clouds: Scientific computing as a service, in: Complex, Intelligent, and Software Intensive Systems (CISIS), 2013 Seventh International Conference on, IEEE, 2013, pp. 481–486.

[30] P. Harsh, Y. Jegou, R. G. Cascella, C. Morin, Contrail virtual execution platform challenges in being part of a cloud federation, in: Towards a Service-Based Internet, Springer, 2011, pp. 50–61.

[31] Rightscale, (accessed August 13rd 2014).
URL http://www.rightscale.com

[32] Amazon elastic beanstalk, (accessed August 13rd 2014).
URL https://aws.amazon.com/elasticbeanstalk/

[33] Cloudfoundry, (accessed August 13rd 2014).
URL http://cloudfoundry.org