Porting Grid Applications to the Cloud with Schlouder

Etienne Michon*, Julien Gossa*, Stéphane Genaud*, Marc Frincu* and Alexandre Burel†
*ICube, University of Strasbourg, CNRS, France
†LSBMO, Institut Pluridisciplinaire Hubert Curien, CNRS, Strasbourg, France

***Abstract*—This paper presents Schlouder, a broker of IaaS cloud resources which helps users to take advantage of IaaS elasticity. The main advantages of Schlouder are its simplicity, its extensibility, and its capability to provide the user with a prediction of the makespan and cost, given the chosen provisioning strategy and before any actual execution. This paper illustrates how Schlouder enables, with a very limited engineering effort, the port of a bag-of-tasks scientific application which was originally developed for the European Grid Infrastructure. Experiments in real environment support assessments on Schlouder efficiency and comparison between grids and clouds for scientific computations. We conclude that porting grid applications to the cloud represents a shift in the associated problematics: from tailoring the application to the platform, to tailoring the platform to the application.**

***Index Terms*—IaaS, EGI Grid, Cloud Broker, OMSSA, Schlouder.**

I. Introduction

Over the past few decades, the scientific community has developed a wide range of solutions to address the problem of ever growing needs for computational power. Among these solutions, grids have shown to be well adapted for computations made of independent tasks, often referred to as *bags-of-tasks* (BoT). Grids offer an efficient paradigm to mutualize a set of computational resources between a number of organizations, thereby increasing the global usage of the resources. The European Grid Infrastructure (EGI) is a prominent example of a Grid environment funded as a sustainable production infrastructure, used by tens of thousands of scientists all over the world. Recently, the concept of Cloud computing has emerged and spread very widely in almost every field of information technologies. Thanks to virtualization, it offers new perspectives in terms of software deployment in distributed systems, and can naturally be seen as an alternative to grids for scientific applications. By contrast with the grid, which incurs mutualized costs, IaaS providers invoice the users solely for the time they have used the resource. The usage is actually accounted using Billing Time Unit (BTU). The BTU often represents an hour, following the pioneer Amazon *on-demand* offer. There are further differences that clearly differentiate grids and clouds. First, Infrastructure as a Service (IaaS) clouds offer a total control over the software stack to users whose can fully customize the VM image. On the grid, a user relies on the responsiveness of a site administrator to install new software. Second, an IaaS cloud is inherently more homogeneous as the type of physical resources is explicitly requested by the user, and the usual practice is to reserve CPUs from the same datacenter. In contrast, granted resources on the grid are of any type by default and despite the possibility to specify constraints on resource properties, the distribution of assigned resource generally differs from one execution to the other. The distributed nature of the grid also applies to storage. Grid storage elements are distributed at a large scale while in IaaS, the storage is provided within the datacenter.

In this paper, we focus on a grid application in the field of proteomics, targeting data interpretation using the Open Mass Spectrometry Search Algorithm (OMSSA) [1] open-source software. The proteomists and computer scientists of the Hubert Curien Pluridisciplinary Institute in Strasbourg have interfaced OMSSA with a production pipeline called *Mass Spectrometry Data Analysis* (MSDA), accessed through a web portal. For its users, such a tool automatizes the parallel execution of several OMSSA instances distributed over EGI in the Biomed virtual organization. This results in a higher analysis throughput and allows for large scale high throughput proteomics projects. We describe in the paper how this mature, grid-based, production environment has been ported to an experimental private IaaS, using a cloud-broker named Schlouder. This task is an occasion to analyze how typical scientific computations are handled in the two environments, by pointing out some essential differences in the light of this experiment. Our analysis, although limited to one application, illustrates through a concrete example the conclusions drawn by broader comparison works, which highlight the complexity of Grids [2].

The issue related to adapt distributed applications from a grid to a cloud environment represents a widely shared concern in the scientific community. In the field of proteomics, researchers started to study how OMSSA executions could be distributed using this IaaS infrastructure shortly after Amazon's EC2 offer became available. Halligan et al. [3] came up with ViPDAC (Virtual Proteomics Data Analysis Cluster), a pipeline comparable to MSDA. This was however more a proof-of-concept framework than an adaption of an existing production environment since one important characteristic of the system was that it did not offer automatic provisioning of the VMs, the user being responsible for setting up manually the number of started VMs to reach the desired speed-up. Manually provisioning the resources is the normal way to operate an IaaS, whereas this operation has great impact on the cost and duration of the computation. This responsibility should not be left to the end user.

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

On the contrary, in our work we aim at providing users the same interface as the one they usually work with, i.e submitting a job with `job-submit` or equivalent. It means we must interpose some interface between the IaaS and the user, whose role is to handle and submit jobs to one (or several) clouds on behalf of the user, exactly as the meta-schedulers do in production grid systems. The contribution of this work is therefore a cloud broker:

- which can easily replace a grid meta-scheduler with the capability to drive the provisioning and scheduling towards performance or cost saving;
- able to predict the execution behavior through a simulation module;
- evaluated on a test-case provided by experimenters in proteomics, compared to the executions carried out on the production grid.

This also demonstrates the ease of porting the application from the grid and the wider range of possibilities we have in terms of resource management. While the grid pipeline focuses on the granularity of the data distribution without caring for the kind of resources used, our cloud broker enables to finely discriminate between which, and when to use the resources.

The paper is organized as follows. Section II describes the environment of the production grid application and Section III describes the architecture of Schlouder, our cloud broker. Then, we compare the results on the grid and on the cloud in Section IV. Finally, we present some related work in Section V.

II. The Applicative Case on the Grid

Let us describe the current production environment and characterize its behavior through typical use-cases in proteomics. 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 for their identification. Each spectrum is then searched against a database of proteins. As each search is independent from the others, a natural parallelization consists in making this application a BoT. The MSDA web portal wraps the different tasks in a workflow that manages data as well as parallel computations. It first copies the appropriate databases to a storage server, then groups the spectra into files to be sent to different computation sites. 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 by MSDA as a function of the requested resolution and the number of available CPUs. One "good" granularity has been determined empirically by the proteomists as 1,250 spectra par task, this granularity yielding overall the shortest makespan on EGI.

Once the jobs are prepared, MSDA submits them to a meta-scheduler hosted locally. The one used in our experiments is JJS [4]. JJS was chosen for its performance in the submission process: after a training phase, all grid sites are permanently ranked depending on their response delays and only the best ones are requested. JJS regulates the submission of jobs by proceeding in rounds to the local resource management systems (LRMS) (i.e. the local schedulers at the remote sites, typically Slurm, SGE, PBS, ...), and then monitoring the progress of previously submitted jobs. Empirical observations of the proteomists led to a size of 50 jobs per round.

III. Schlouder: A Cloud-Broker Alternative

We have ported the very same experiment on a cloud infrastructure, using our cloud-broker called Schlouder[1]. On one hand, Schlouder provides a user interface to submit jobs in a very similar way to any batch-scheduler. On the other hand, it interfaces with an actual batch scheduler and a cloud resource manager. Both interfaces are very modular and can be adapted to any specific solution, particularly regarding the cloud interface which can support several concomitant modules in order to manage cloud federation. 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 preferences.

A. Schlouder's Architecture

The main functions of Schlouder are (1) to decide when to start or shutdown VMs, and when to reuse already running ones according to a given provisioning strategy, and (2) to execute the submitted jobs through the batch scheduler interface. The only specific requirement is to embed a batch-scheduler node into the VM images.

Figure 1 depicts the architecture of Schlouder and the typical use-case:

- 0.1, 1.1 : A user submits a job along with its duration through the Schlouder client, which communicates with Schlouder on the frontend.
- 0.2–0.3: If requested by the user, Schlouder calls the Provisioning Simulation Module (PSM) to provide the user with an estimation of the cost and the makespan of the job execution using all available strategies. It does not actually provision resources on the platform;
- 2.1–2.3: Schlouder's provisioning module decides how the infrastructure should be scaled and instructs the cloud kit (Eucalyptus and OpenStack are supported) accordingly. If a VM instance is started, a Slurm agent embedded in the VM image starts at boot time and notifies its existence to the Slurm controller. Hence, the scheduling system becomes aware of the new available VM;
- 3.1–3.2: Schlouder schedules the jobs for execution on the VMs through the Slurm controller.

1) Schlouder Engine: The Schlouder engine is the conductor of the broker. When it receives submissions from the client, it calls the different modules and interacts with the third party software (batch scheduler and cloud kit). It applies the requested provisioning strategy, and monitors both the

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

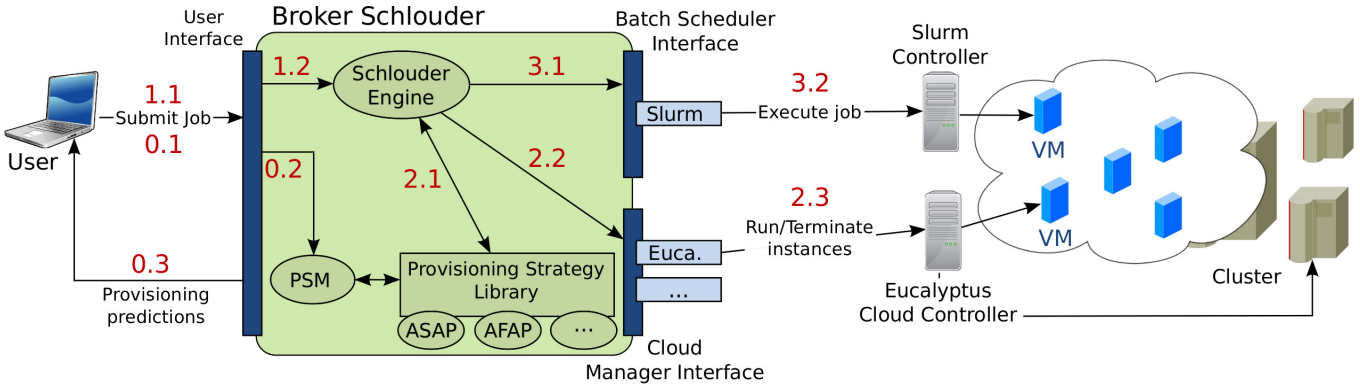

Fig. 1. The architecture of Schlouder

virtual platform and the execution. The Schlouder engine can also predict the VM startup time which is important for the provisioning strategy. It is easy to predict it on most public clouds as it is almost constant [5], and can be modeled as a linear function in our private cloud (cf. Section IV-A3).

2) Provisioning Strategy Library: In [6] we designed and assessed a dozen provisioning strategies. However, it is out of the scope of this paper to investigate the results of each strategy. Our objective here is mainly to demonstrate that well differentiated strategies must be proposed to cloud users. We will therefore only use the following two strategies:

- *As soon as possible (ASAP)* minimizes the wait-time, regardless of the renting cost, by deploying a VM for each job that can not be handled immediately by already provisioned VMs, or when the boot time exceeds the wait time for one VM to become available.
- *As full as possible (AFAP)* aims at filling the BTU as much as possible, hence sparing the number of rented resources by delaying the execution of the submitted jobs.

In case the maximum number of VMs is reached, pending jobs are assigned to the first VM to become idle.

Schlouder offers an open framework to describe any provisioning strategy. Each strategy takes the form of a Perl class using a simple API to access the characteristics and state of both the virtual platform and jobs. The available strategies are stored into a dedicated library, and can be used independently for each job.

3) Provision Simulation Module: Even if the client is helped by a broker that proposes several strategies, choosing one of the strategies is easier when the resulting cost and makespan can be estimated. We have shown that it is difficult to *analytically* estimate the outcome of provisioning strategies in online scheduling [6], due to ceiling effects induced by BTUs [7]. However, we can simulate the strategies using adapted simulation tools to predict their results. That is why we offer in Schlouder a Provisioning Simulation Module (PSM). This module is built upon the discrete event simulation toolkit SimGrid [8] and can predict the number of BTUs (i.e. the cost) and the makespan for all the available strategies in the provisioning library. This is, to the best of our knowledge, a unique feature compared to related projects.

The PSM input is twofold: the target workload (i.e. duration and communications of its tasks) provided by the Schlouder engine, and a description of the platform. The description follows the SimGrid formalism and XML file format. Plug-and-play platform files, such as Amazon EC2, are already available. Its main output is the cost and makespan for each available provisioning strategy. However it can also provide very precise information about the simulation, which is useful to analyze and improve provisioning strategies offline.

B. Schlouder in the cloud stack

The cloud computing ecosystem comprises multiple layers. IaaS and Platform as a Service (PaaS) are just two examples suited for developing and running custom applications. Each layer hides more components managed in the previous one by the client. 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. However, unlike PaaS, it does not impose one development platform or tools, completely isolates clients, and allows them to manage their own customized VM images.

C. Porting the application to the cloud with Schlouder

In order to port the application with Schlouder, one must (1) embed the executable into one VM image, (2) adapt the grid submission scripts to change the way data are handled, and (3) upload the data to the cloud storage. As the input data may vary from one execution to another, the latest step is the only one to be done for each execution. The calibration made for the grid is kept (i.e. spectra are still grouped by 1,250 as it was set for the grid). These three steps were done within an hour. To submit a job to Schlouder, a user would simply type in the following command:

```
schlouder-client [--psm] <executable> <args>
```

where the `--psm` option instructs the client to call the PSM instead of making an actual submission to the compute nodes.

IV. Evaluation

The evaluation aims at assessing Schlouder efficiency in regard to the original production environment based on EGI. The discussion is supported by the observations of MSDA

runs over two months. For sake of comparison, the exact same workload has been replayed on our private cloud with Schlouder.

A. Experimental setup

1) Test-case: The proteomists have selected as reference four representative use-cases 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*), low-resolution measurements and full trypsin cleavage (*lrt*), and finally low-resolution measurements and semi-trypsin cleavage (*lrs*). The high-resolution measurements contain 65 jobs for 58 MB of input files, and the low-resolution measurements contain 223 jobs for 548 MB of input files.

The job duration is almost linear in the number of spectra in the input file but highly depends on the search type. The prediction can be accurately modeled using linear regression, with correlation coefficients of 0.903, 0.986, 0.985 and 0.970 for *lrs*, *lrt*, *hrs* and *hrt* respectively.

2) Grid: It is difficult to find out the precise aggregated computational power used in grid-based executions since each run may be assigned a different set of resources. We noticed that most worker nodes were running on recent Intel and AMD processors, such as AMD K10-based Opterons and Intel Nehalem-based Xeons. Regarding storage, the database used for peptide identification and the OMSSA executables are stored in a Storage Resource Manager at a single location. Once an EGI site has been chosen by JJS, the database and the executables are transferred to the site's worker nodes [2].

3) Cloud: Our private cloud setup is composed of 3 local servers, each being a dual $2.67GHz$ Intel Xeon X5650 processor, with 24 hyper-threaded cores. Hence, we can run up to 72 single core VMs. 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). All machines from our IaaS run on a Linux 3.2.0-26 with the KVM module. For this experiment, we used Eucalyptus 2.0.3 [9] as cloud kit and Slurm [10] for batch scheduling.

The startup time, which is optimized by Eucalyptus with a VM image cache mechanism, is almost linear in terms of the number of booted VMs. We observed that the `run_instances(`n`)` command, with $n \in [10; 60]$, by step 10, was correctly approximated by a linear regression $boot\_time = 5.55i + 50.96$ (in s), which predicts when the ith VM is booted. The correlation coefficient is 0.993.

B. Observations

The differences in behavior between the two systems are well highlighted by analyzing the breakdown of the overall execution time (named *walltime* in the following). From the analysis of the execution logs, we have identified 5 time lapses of interest, able to characterize the grid execution behavior. They will be the basis for our comparison with executions on the IaaS. Figure 2 shows the life cycle of a job execution, from the time (t_0) a job is admitted to the meta-scheduler queue to its completion (t_4) at the production site. At t_1 the meta-scheduler submits the job at the LRMS of the selected site. As errors are frequent on the grid, the figure exemplifies such an error (e.g., the selected site does not respond), which is detected at time e. The error time is thus defined as $e-t_0$. The job is immediately queued for re-submission during a period $t'_1-t'_0$ called wait time, and delivered to another site at t'_1. In the cloud setup, the wait time starts when the job is submitted to Schlouder and stops when Schlouder is able to direct the job (through Slurm) to the VM of execution. Once at the site, the job is queued on the site's LRMS during $t_2-t'_1$ (submission time). The job actually starts at t_2, with a first phase consisting in downloading the database and the executable, and a second phase represented by the OMSSA computations.

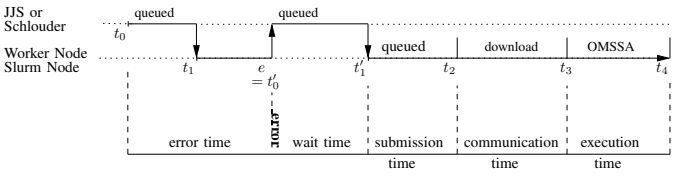

Fig. 2. An example job's life-cycle showing the times observed in experiment.

The breakdown of the walltime (in s) is reported in Table I. This is an average over 4 to 10 runs for each search type (*hrs*, *hrt*, *lrs*, *lrt*). Columns *ASAP* and *AFAP* refer to the strategies used for executions on the IaaS cloud.

Another view of the behavior is the evolution in the number of concurrent jobs and VMs running during each execution. Figure 3 presents how the platform's diameter (i.e the number of concurrent jobs or VMs) evolves as a function of time. We will refer to this figure to explain certain observations in the following.

C. Comparison of Grid and Cloud-ASAP

Let us first compare the behavior of our sample runs on the grid and on our cloud using *ASAP*. For both cases, the user's wish is to (very classically) complete the execution of its workload the sooner, regardless of the resources consumption.

It comes as no surprise that executions on EGI suffer from transient system errors. However, the number of errors is striking, especially in comparison to the IaaS which never failed in this experiment. The time spent in error is about 30% on average of all runs. We even noticed jobs that failed several times and the *hrs* test shows more failed jobs than successful jobs. The time wasted because of errors is however not correlated to the number of errors since jobs are submitted in parallel. The causes of the errors are various: aborted computations due to timeout, middleware failure, refused connection, physical failure, etc. Anyhow, the widely distributed and multi-administrated nature of the grid makes errors difficult to identify and correct. Most of the time, error detection is based on timeouts, which explains the large overhead incurred by errors to grid submissions. This can even be seen on some of the plots of Figure 3: for *hrs* and *hrt* with

[2]For reproducibility, the OMSSA configuration files are available at http://schlouder.gforge.inria.fr/omssa_configuration/

TABLE I
BREAKDOWN OF THE WALLTIME OF THE GRID AND CLOUD EXPERIMENTS

	hrs (65 jobs)			*hrt* (65 jobs)		
	Grid	ASAP	AFAP	Grid	ASAP	AFAP
(1) Error time (#Jobs in error/#Jobs)	34.36 (127.6/65)	0	0	240.69 (27.25/65)	0	0
(2) Wait time	158.53	259.28	1530.40	159.29	103.20	313.75
(3) Submission time	49.14	2.18	0.90	110.4	1.19	0.89
(4) Communication time	25.69	0.39	0.22	13.48	0.27	0.22
(5) Execution time	234.15	161.18	135.79	33.50	7.22	6.64
Wall time	1026.80	486.50	3388.00	768.50	142.50	557.20

	lrs (223 jobs)			*lrt* (223 jobs)		
	Grid	ASAP	AFAP	Grid	ASAP	AFAP
(1) Error time (#Jobs in error/#Jobs)	17.19 (195.88/223)	0	0	87.11 (55/223)	0	0
(2) Wait time	239.27	1191.40	1803.00	67.22	209.71	1476.50
(3) Submission time	72.06	1.36	0.97	157.43	1.20	0.90
(4) Communication time	26.98	0.53	0.28	2.06	0.35	0.26
(5) Execution time	816.80	855.67	571.08	118.74	24.32	20.56
Wall time	2438.50	3373.33	4251.60	887.00	419.25	3505.75

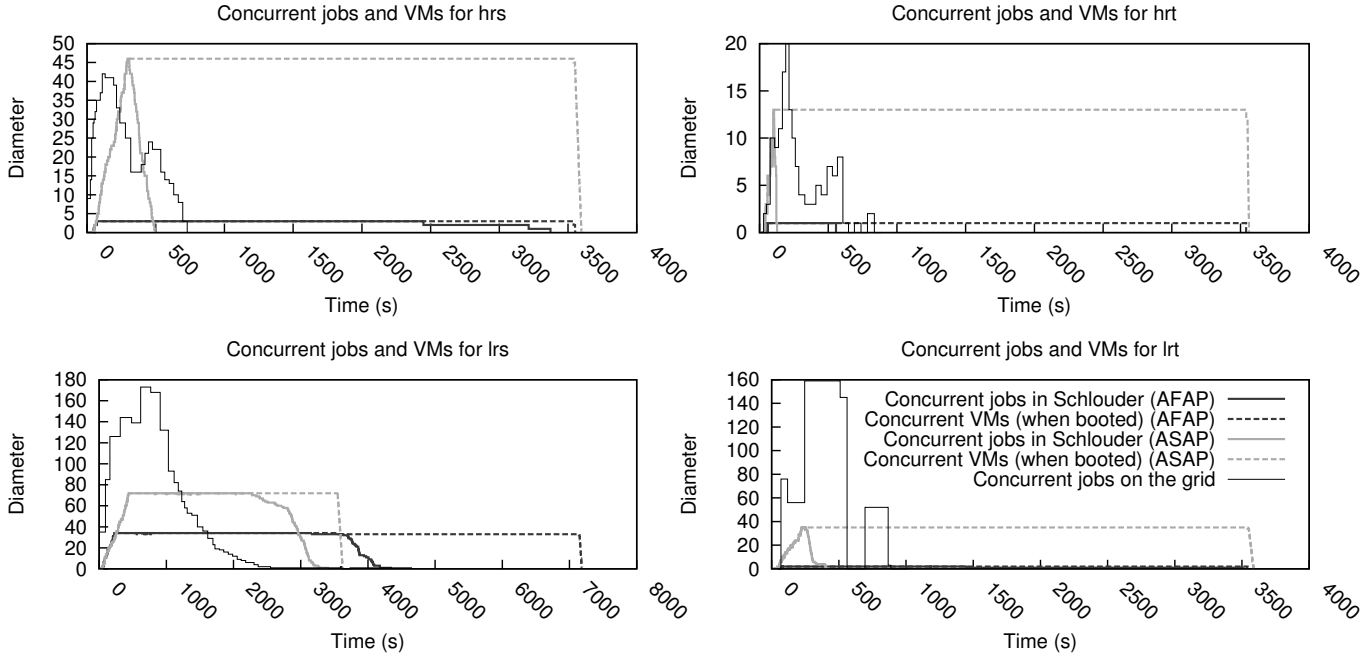

Fig. 3. Concurrent jobs and VMs on the cloud and on the grid

65 jobs, after the first wave of jobs starting concurrently, we see a second wave starting at about 300s, which are the jobs resubmitted after a timeout.

Wait time tells us how long it takes for a job to be admitted on the computing site. Although wait times presented in Table I are similar in the two environments, the nature of the wait differs. In the Grid experiment, wait time happens at the submission site because the meta-scheduler JJS cannot instantaneously dispatch the jobs to the remote sites. With Schlouder, two factors participate in the wait time: i) the delay to boot a VM, and ii) the limited number of VMs that can be started (72). The former factor is controlled by Schlouder, which estimates whether starting a new VM or waiting for a VM to be available is best to execute a job the soonest. For instance, the cases *lrt* and *hrt* (trypsin searches) involve short jobs, many of which can be executed on a same VM in sequence. Hence, the number of started VMs is limited in comparison with the 223 jobs to execute. The latter factor is proper to our private IaaS: once 72 VMs have been started, the jobs have to wait for one of the VM to be available (we execute jobs one at a time). By contrast, EGI resources are never exhausted so that this penalty never happens in Grid runs. It clearly appears that while such wait time cannot be avoided on grids, those observed on the IaaS cloud represent an overstatement of the typical behavior. Indeed, the wait times would be largely reduced if the platform was in a steady-state, i.e a number of VMs are already running when jobs start arriving. Notice that the system can be brought into steady-state by a simple pre-provisioning strategy.

The submission time can be seen as a measure of the computing environment reactivity. It is the delay between the job's admission at the LRMS and its actual execution. On

the EGI Grid, the LRMS is represented by an access service (LCG-CE or CREAM) which in turn connects to a site specific job-scheduler (we observed the use of PBS, LSF, SGE, and BQS throughout our experiments). With Schlouder, it is the time the job stays in Slurm's queue. It is not surprising to see longer delays on the grid given the hierarchy of levels to access the final worker node. On one hand, grid middleware handles a complex resource management (security, scheduling, claiming, monitoring, ...), and on the other hand using an IaaS enables to build a lighter execution environment embedded into the VMs, that can be precisely tailored to its user's needs. Communication times are also slightly greater on the grid than on the cloud in this experiment even though the data transfers are small. This reflects another essential difference between grids and clouds: The distributed nature of the grid implies long-distance communications as files need to be uploaded on each remote site. On the contrary, clouds offer centralized storage, through a local distributed file system. Thus, as soon as only one site is used, communications on IaaS cloud are on a LAN, making it more suitable for data-intensive applications.

The execution times we report for EGI and the IaaS are not directly comparable because of the difference in the processors used, and because the logs from the grid application only show the start and end times as reported by the LRMS. It is clear that the measures reported from the grid also account for some startup time per job since the ratio to the execution times on the cloud are large (about 5 times) when the experiment deals with small jobs (*hrt* and *lrt*) and smaller (about 1.5 times) with longer jobs. Nonetheless, the execution times observed with the *AFAP* strategy on the cloud also include some overhead, as will be seen in next section, and overall do not really differentiate the two environments.

D. Comparison of Cloud-ASAP and Cloud-AFAP.

Let us now examine how using opposite strategies (see section III-A2) change the execution behavior. Figure 3 shows how both strategies provision resources to reach their respective objectives. On one hand, *ASAP* provisions a new VM for each job, as long as it predicts that the resulting gain in performance will overcome the boot time penalty of this VM. The number of provisioned VMs grows up to 13 for 65 jobs in *hrt* case, and up to 72 for 223 jobs for *lrs*. However, most of the rented BTUs remain unused, as the dotted line representing the number of concurrent VMs shows. For instance, such executions on Amazon EC2 would have cost from 0.78$ to 8.64$. On the other hand, *AFAP* postpones job executions in order to occupy most of the BTUs created by each started VM. Thus, the number of started VM is largely reduced as compared to *ASAP* (from 1 for *hrt* to 34 for *lrs*). For instance, such executions on Amazon EC2 would have cost 0.06$ to 2.04$, substantially sparing from 76% to 92% of the expense. Naturally, the counterpart is the increase in the wait times (from 51% for *lrs* to 604% for *lrt*).

The average submission and execution times per job are however noticeably longer with *ASAP* than with *AFAP* (up to 1.4 times longer in the worst case for the execution time). This reveals a pitfall for strategies based on immediate reaction to a workload peak: the system undergoes an overload due the large number of VMs booting simultaneously. The same applies to a smaller extent to the batch scheduler Slurm, explaining the longer submission times due to the overload. Consequently, even though the users rent homogeneous cloud resources, their performances depend on the usage of the physical machine, which might depend on other users and can not be predicted nor monitored by cloud end-users. This is currently a severe limitation to the accuracy of prediction models in multi-tenancy systems.

The main weakness of the *AFAP* strategy is its sensitivity to runtime predictions. These predictions are used to determine up to when jobs should be scheduled to a same BTU so as to minimize cost. We observe in the *lrs* case that the prediction underestimated the execution times and lead to consume an extra-BTU, wastefully doubling the cost of the execution. However, such a worst case can be avoided by overestimating the execution times.

E. Accuracy of the Provisioning Simulation Module

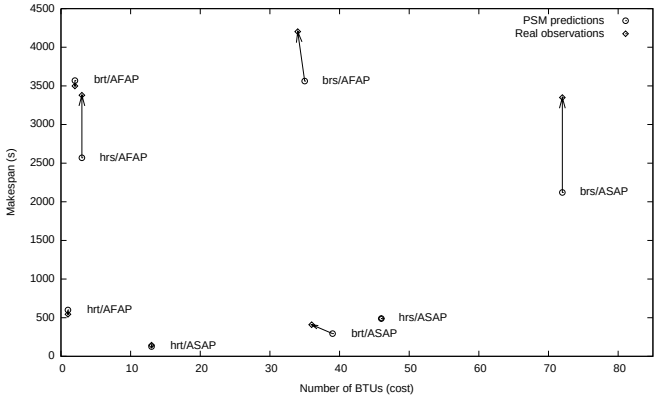

Fig. 4. PSM predictions and real outcomes for cost and makespan.

The PSM has been configured to simulate our platform of experiments. Figure 4 compares the predictions by the PSM and the real executions for the four workloads and the two provisioning strategies. Each PSM execution took between 0.5 s and 1.2 s. Thus the PSM does not present a performance issue.

A first observation is that the simulation is very accurate in most cases. The PSM estimates the makespan with an error ratio ranging from 0% to 36%, and the costs are assessed with an error ratio ranging from 0% to 8%. Most of the error comes from the prediction inaccuracies of the execution time. Indeed, this inaccuracy for *lrs* and *hrs* ranges from 2 to 343 seconds (3% to 42% of the runtime), while it does not exceed 4 seconds (maximum 22% of the runtime) otherwise. We have evaluated the PSM accuracy with the actual runtimes instead of predictions. This last experiment shown that the PSM almost perfectly assesses the outcomes when the runtimes can be perfectly defined.

The PSM provides the user with globally accurate outcomes, especially regarding the ranking of provisioning strategy alternatives. Thus, it allows the user to decide which strategy to

use, with all information in hand. Finally, the PSM[3] has been designed to be easily adaptable to any cloud infrastructure and any provisioning strategy.

F. Conclusion

In this study, our cloud framework has shown to be a suitable alternative to the grid. For scientists, Schlouder allows to benefit from the cloud advantages: it is elastic, cheap and reliable. Moreover, the cloud offers a trade-off between cost and performance. Technically, the cloud does not need all the middleware complexity of the grid, sparing concomitant overheads and latencies.

Furthermore, our study shown that Schlouder allows to drastically reduce the experiment setup phase. Indeed, setting this experiment up on the grid needed to develop a dedicated submission portal (MSDA) able to monitor the execution, to use a dedicated meta-scheduler (JJS) specifically trained to rank the available resources, and to test different configurations, both in term of batch size and job size. On the other hand, setting the experiment up on Schlouder only needed to embed the executable into a VM image and slightly adapt the grid scripts to use the cloud storage. This was done within an hour, with no further calibration nor deep knowledge of the execution platform, while the application is now totally portable to any cloud infrastructure, and the execution is adaptable to different user's preferences.

However, we highlighted that performances of the VMs are greatly impacted by the usage of other users, which can not be known on clouds. Furthermore, large-grained leasing time through BTU implies to carefully manage the resource provisioning, at the risk of wasting money. This implies to have accurate information on the runtime of the jobs, which can prove to be difficult on certain applications. Finally, we provide an accurate way to estimate the way a workload would be handled by giving an assessment of the outcomes of the provisioning strategy alternatives. Nevertheless, the grid is competitive despite the high number of errors we observed, Schlouder proves to be a much easier way to run scientific computation, and can be used to port existing applications with almost no effort.

V. Related Work

The problem of provisioning the appropriate amount of cloud resources when cost and makespan are considered has been extensively studied. Many algorithms and multi-cloud platforms (at IaaS and PaaS level) have been proposed in the recent years. In this section, we focus on their characteristics as the algorithms' efficiency has been already addressed in [6]. IaaS and PaaS systems proposed both by academia and commercial company are investigated. We were particularly interested in their ability to provide clients with customized provisioning and scheduling strategies and recommendation systems in terms of their efficiency; whether or not they restrict users to an API or programming model; and their availability to the open source community. We included in our comparison solutions offering brokers and single cloud platforms. Interestingly most brokers lie at IaaS level as they require 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 began adding intercloud capabilities. We divide the related work in two parts: (1) PaaS with automatic provisioning and (2) client oriented IaaS brokers.

A. PaaS solutions with automatic provisioning

Projects like Aneka [11], mOSAIC [12] or COMP Superscalar [13] together with many commercial PaaS (e.g., Google App Engine) 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 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. [13] propose a platform and a programming model based on Java annotation to execute an application on an IaaS cloud. Vecchiola et al. [11] and Petcu et al. [12] 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.

B. Client oriented IaaS brokers

STRATOS [14] and Kingfisher [15] are examples of frameworks dealing with intercloud 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. 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 30s.

Brokers for general scientific computations have been investigated in [16], [17], and [18]. The main difference with our solution for the following two works is that they do not provide a method of customizing the provisioning policy. In [16] a cluster architecture to deliver flexible and elastic High Throughput Computing environments is presented. The aim is to grow a local cluster's capacity using an external cloud provider. The results demonstrate that only the overheads due to virtualization affect the performance of the elastic cluster.

[3]This work is in progress, and will be submitted to publication soon.

TorqueCloud [17] 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.

Another tool, aimed at testing user-defined strategies, has been developed by Villegas et al. [18]. Their work is the closest to ours. The platform 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 their solution is not available for download and testing, its main issue is the lack of recommendation system like PSM.

Among commercial brokering systems, one of the most well-known is RightScale 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, CloudFoundry [19] or Windows Azure provide similar services but without multicloud support.

VI. Conclusion

This paper has presented Schlouder, a broker designed to support scientific computation on the cloud. Its usage has been illustrated through the port of a real application, that is a high throughput proteomics data interpretation. Schlouder have been compared to the initial production software environment developed for the exploitation of on the European Grid Infrastructure (EGI), and proved to be one suitable alternative.

The future work is twofold. 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 PSM. 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 continue to take benefit from the virtual homogeneity of cloud resources by designing a P2P monitoring module, able to share information about both public cloud infrastructures and application runtimes among the Schlouder users.

To conclude, the issues to be addressed regarding resource management are very different in grids and clouds: from discovering, allocating, and monitoring grid resources, to provisioning and scheduling cloud resources. Indeed, while the main efforts on the grid aim at experimentally organizing the workload and finding the good submission pattern to exploit as many resources as possible, the main effort on the cloud is to define how many resources must be provisioned given user preferences. As this issue is already handled by Schlouder, grid applications can be ported to the cloud with no further effort. From tailoring the application to the platform, to tailoring the platform to the application: the cloud represent a major shift of the scientific computing problematics.

References

[1] L. Y. Geer, S. P. Markey, J. A. Kowalak, L. Wagner, M. X. D. M. Maynard, X. Yang, W. Shi, and S. H. Bryant, "Open mass spectrometry search algorithm," *J Proteome Res.*, vol. 3, no. 5, pp. 958–964, 2004.

[2] S. Jha, A. Merzky, and G. Fox, "Using clouds to provide grids with higher levels of abstraction and explicit support for usage modes," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 8, pp. 1087–1108, 2009.

[3] B. D. Halligan, J. F. Geiger, A. K. Vallejos, A. S. Greene, and S. N. Twigger, "Low cost, scalable proteomics data analysis using amazon's cloud computing services and open source search algorithms," *Journal of Proteome Research*, no. 8, pp. 3148–3153, 2009.

[4] P. Calvat, T. Glatard, S. Camarasu-Pop, C. Chuen Teck Mark, and T. Wen-Jun, "Two experiments with application-level quality of service on the EGEE grid," in *Proceeding of the 2nd workshop on Grids meets autonomic computing*, Jun. 2010.

[5] M. Mao and 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.

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

[7] E. Michon, J. Gossa, and 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.

[8] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: a Generic Framework for Large-Scale Distributed Experiments," in *10th IEEE International Conference on Computer Modeling and Simulation*, Mar. 2008.

[9] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2009, pp. 124–131.

[10] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *9th Intl Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, 2003, pp. 44–60.

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

[12] 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*, vol. 2, no. 1, p. 12, 2013.

[13] E. Tejedor, J. Ejarque, F. Lordan, R. Rafanell, J. Alvarez, D. Lezzi, R. Sirvent, and 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.

[14] P. Pawluk, B. Simmons, M. Smit, M. Litoiu, and S. Mankovski, "Introducing stratos: A cloud broker service," in *5th International Conference on Cloud Computing (CLOUD)*, Jun. 2012, pp. 891 –898.

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

[16] R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "An elasticity model for high throughput computing clusters," *Journal of Parallel and Distributed Computing*, vol. 71, no. 6, pp. 750–757, Jun. 2011.

[17] H. Song, J. Li, and X. Liu, "IdleCached: an idle resource cached dynamic scheduling algorithm in cloud computing," in *9th International Conference on Ubiquitous Intelligence and Computing (UIC) and Autonomic and Trusted Computing (ATC)*, Sep. 2012, pp. 912 –917.

[18] D. Villegas, A. Antoniou, S. Sadjadi, and A. Iosup, "An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds," in *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 2012, pp. 612–619.

[19] "Cloudfoundry," (accessed July 17th 2013). [Online]. Available: http://www.cloudfoundry.com/