Computing and Informatics, Vol. 31, 2012, 119–134

TRANSPARENT ACCESS TO SCIENTIFIC AND COMMERCIAL CLOUDS FROM THE KEPLER WORKFLOW ENGINE

Marcus Hardt, Thomas Jejkal

KIT, Karlsruhe Institute of Technology
Kaiserstrasse 12
76131 Karlsruhe, Germany
e-mail: hardt@kit.edu

Isabel Campos, Enol Fernandez

CSIC, Spain

Adrian Jackson, Michele Weiland

EPCC, United Kingdom

Daniel Nielsson

Chalmers, Sweden

Bartek Palak, Marcin Płociennik

PSNC, Poland

Abstract. This paper describes the architecture for transparently using several different Cloud Resources from with the graphical Kepler Worklfow environment. This architecture was proven to work by implementing and using it in practice within the

FP7 EUFORIA project. The clouds supported are the Open Source cloud OpenNEbula (ONE) environment and the commercial Amazon Elastic Compute Cloud (EC2). Subsequently, these clouds are compared regarding their cost-effectiveness, which covers a performance examination but also the comparison of the commercial against a scientific cloud provider.

Keywords: Cloud, Amazon, EC2, AWS, Kepler, workflow, OpenNEbula, ONE

1 INTRODUCTION

This paper is based on work done within the EUFORIA project, which is funded through the Research Infrastructures initiative of the 7th Framework Programme of the European Commission, grant agreement No. 211804.

The goal of the work presented in this paper was to enable end-user scientists to easily allocate enough computers to run their code on. It was therefore mandatory to hide the details of the cloud from the user. This paper describes our solution which accomplishes this.

These considerations lead to the design and implementation of an architecture which integrates cloud resources to be available for the computations of our users. The designed architecture includes access to the cloud resources via the graphical workflow engine Kepler [4].

Resources in terms of this paper are those provided by cloud infrastructure providers (IAAS) such as Amazon Web Services (AWS). Higher level cloud services (e.g. PAAS, SAAS) where not within scope.

A second goal of this work was to find out whether it is financially more attractive to run commercial cloud services in a scientific computer centre or to use commercial cloud offerings. We have therefore conducted a cost comparison between Amazon EC2 and a large scientific compute centre. Furthermore, benchmarks have been used to judge the difference in performance between both providers.

This paper starts with an introductory evaluation in Section 2 of cloud infrastructures, before the architecture is described in detail in Section 3. Section 4 analyses in detail the cost that emerge when running a cloud centre and compares the obtained figures with the prices charged by Amazon. The paper finishes with a brief evaluation on benchmark performance in Section 5.

2 EVALUATION OF CLOUD INFRASTRUCTURES

The recommendation given to EUFORIA [3] was targeted at trying out an Infrastructure as a Service (IaaS) provider, because platform and software (PaaS and SaaS) are likely too high-level for being integrated on the same level as grid and HPC.

Choosing the cloud provider was not very easy, as many commercial cloud providers exist. Typically each provider requires the user to learn and use their specific interface. This is typically a web GUI which is often supplemented with a vendor specific application programmers interface (API).

Implementing against such an API leads to the so called "vendor lock-in", which describes the situation that the user has implemented his access to the provider so specific that s/he cannot change the provider, without reinvesting in a re-implementation of his/her software. To avoid this, all vendors would have to agree upon one standardised interface. Among others, the UK based ElasticHosts (`http://elastichosts.com`) was actively pursuing the standardisation of the open, OGF-standardised Open Cloud Computing Interface (OCCI) (`http://occi-wg.org`) protocol. OCCI is also supported by the OpenSource OpenNEbula (ONE) (`http://opennebula.org`) [1] cloud middleware.

However, any serious cloud implementation should be comparable to the offerings provided by Amazon Web Services (AWS) (`http://aws.amazon.com`). Amazon is the biggest and most present IAAS provider. Thus, it serves as a de-facto standard for cloud computing interfaces. Futhermore, any serious comparison has to be comparable with Amazon.

We therefore decided to take the middle way of using the AWS interface on the one hand and to develop against the provided subset of the AWS interface as supported by an Open Source middleware.

Based on a previous evaluation by KIT of Amazon-compatible OpenSource reimplementations Eucalyptus and OpenNEbula, we have decided to use the AWS interface of OpenNEbula as the reference development platform. ONE is the open-source part of the EC funded RESERVOIR project. Using its AWS interface ensures our implementation can directly be used with Amazon resources. Anticipating one result of the cloud pilot, we can say that our implementation does in fact work seamlessly with either ONE or Amazon resources.

3 ARCHITECTURE

This section details the architecture and the considerations made for the implementation. The architecture was designed with focus on implementing a working solution rather than a theoretical model. Therefore, the implemented technical features are the fundamental ones, leaving space for more advanced ones to be added at a later stage. The overall architecture is therefore stable to work with and extensible for future work.

The overall goal of this architecture is to enable end-user scientists to easily allocate cloud resources that run their code. We have therefore defined the following cycle which describes responsibilities or roles for the respective steps:

1. Admin provides access to a virtual machine to users.
2. Users install their software and configure the virtual machine.

3. Admin defines modified virtual machine image as new master image.
4. User includes the cloud Kepler actor in his workflow. Every time the actor is activated, it triggers all actions necessary to run the code on a newly instantiated VM on the cloud.

3.1 Virtual Machine Considerations

Infrastructure as a Service (IAAS) provides an interface to start virtual machines (VMs) – not more. We had to make choice on how to run the users codes inside the virtual machines. Two possible options exist. The obvious solution is to follow the grid paradigm and to start a cluster of VMs. This cluster can be used with the existing grid interfaces to submit the jobs. The main problem with this "grid on cloud" solution is that the somewhat traditional idea of grid computing would be translated to the cloud. Furthermore, a cluster service does already exist at Amazon and can be included into our approach.

We therefore chose to start a VM in such a way that it does not require further instructions during runtime. We pass a set of parameters to the virtual machine during start-up, including

- the code to run
- parameters
- from where to get input files
- where to store output.

We call this procedure "contextualisation" because it puts the clone of the master VM into the context in which it runs.

The advantages are that no knowledge about the VM is required during runtime and that no network connections need to be made to it for successful execution. For monitoring the VM during development, we have installed a webservice which provides monitoring information when needed. The user can start many VMs with different parameters to run his computations. Currently only serial jobs for use cases like parameter scans are supported.

For the development of the VM, we have created a master-VM, which runs constantly on the ONE cluster at KIT. This machine is provided for developers and code owners to interactively log-in so that the EUFORIA developments and the users' codes can be deployed and tested. When finished, the master VM holds an installation of the codes, libraries and tools required by the users.

This setup enables us to start identical copies of the master VM on the ONE cluster at KIT and at AWS using the same interface, only using different credentials. Currently only one master VM is supported, on which all codes need to be installed. It is easily possible to add several master VMs, each of which can then be custom tailored to the requirements of the specific code. This would be required to utilise Infiniband-enabled cloud clusters with MPI, which are available at commercial providers. This was out of scope for the pilot project, but will be pursued in the future.

At the end of a computation, after the code has stored the output at a specified output location, the VM is shut down.

3.2 Building Blocks

To describe the architecture in more detail, we first need to define a small set of building blocks or services which will be discussed later on.

Kepler workflow orchestrator: Originally designed to run a user's workflow on a local workstation, its capability was extended throughout EUFORIA to submit jobs to external computing resources, such as grid or HPC. Within the cloud pilot project this was extended to instantiate a VM on cloud resources and run the job inside.

Storage Element: This is the place where input files and job outcome are stored. Currently a grid storage element is used for this, which requires to use the grid authentication from within a cloud VM to access storage. Cloud storage will be considered in future works.

Virtual Machines (VMs): All VMs used throughout this paper are based on one master image on which all required software was preinstalled. The master image used for this paper is based on a [5] worker node, but any other VM (including *BSD and Windows) would be possible. Copies of the master node are instantiated on Amazon or OpenNEbula resources.

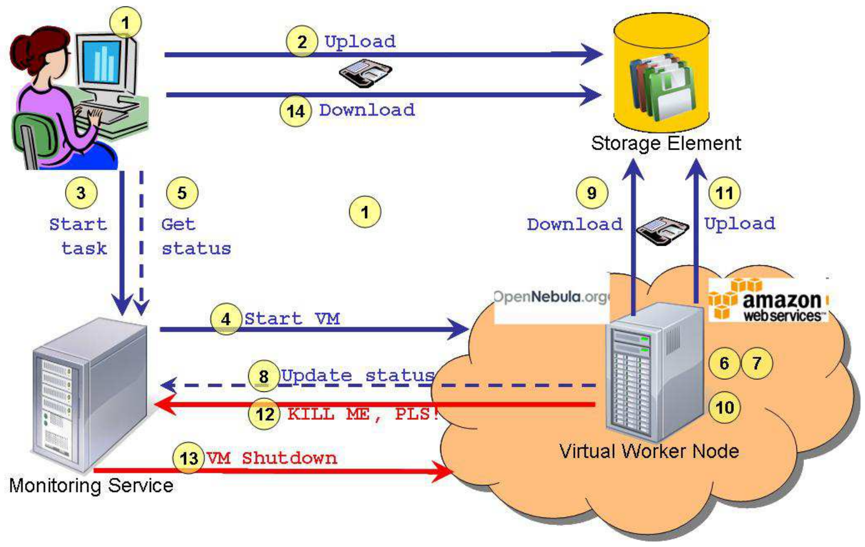
Monitoring Service: Keeps information about submitted jobs and the status of the virtual machines. This is in principle not required and will be omitted in future versions.

3.3 Usage Scenario

The diagram presented in Figure 1 shows the flow of control and data between the system components during the execution of a typical usage scenario from specification of application and input data to obtaining job outcome.

3.4 Monitoring Service

The Monitoring Service is a simple webservice that stores and provides information about jobs, including current job statuses. The service intermediates between Kepler actors which run locally on the user's workstation and the VMs on the cloud infrastructures. This additional link in our call chain is necessary to work around several problems:



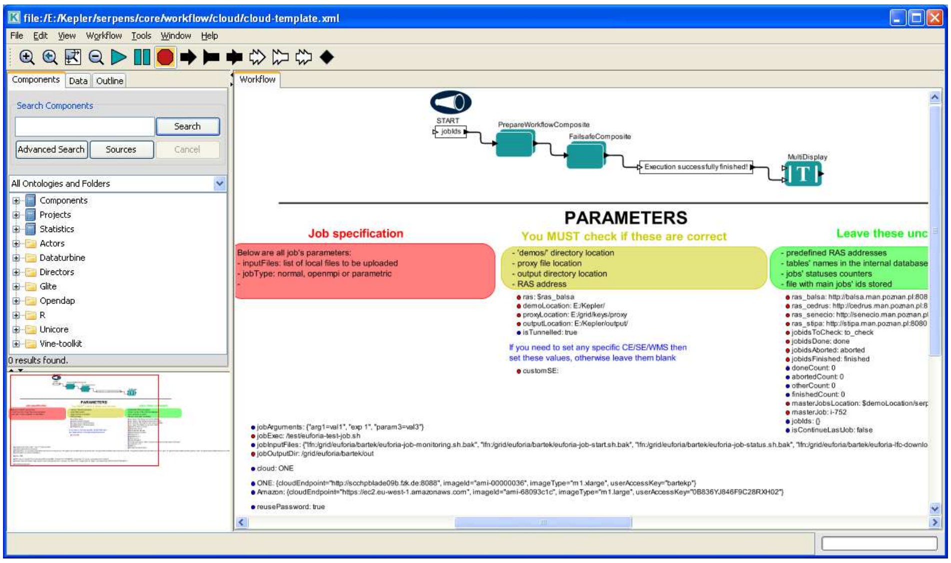
1. User starts a Kepler workflow on the local workstation.
2. Kepler actor uploads job input files to file storage.
3. Request for VM creation is sent by Kepler actor to the Monitoring Service.
4. Monitoring Service passes a request for VM creation to cloud.
5. Kepler actor starts querying Monitoring Service to track job status
6. Virtual Machine is instantiated
7. Pre-installed scripts set up the job environment based on contextualization data
8. VM starts updating job status on Monitoring Service
9. VM downloads application input data
10. Job performs its execution
11. Job output is uploaded to file storage
12. VM calls Monitoring Service passing "KILL ME" signal
13. Monitoring Service shuts down Virtual Machine
14. Kepler actor downloads job outcome from file storage to the local filesystem

Fig. 1. Usage scenario. The numbers 1–14 indicate the order in which the process is traversed.

- Often, scientific compute providers block some ports by firewalls. In this case the Kepler actor cannot communicate with the VM. The VM, however, can still initiate outgoing connections.
- All monitoring information is pushed by VM, so communication is possible regardless of very restrictive rules defined on firewalls blocking access to cloud infrastructure.
- We do not have to keep a list of running cloud instances. Existing machines publish information about their existence. However, for increased robustness we keep such a list.

Storing information about jobs and their statuses in a simple data base allows to shut down VM immediately after job ends or fails. This releases resources (limited by some cloud providers) and reduces costs (running a VM on commercial cloud costs money!).

3.5 Integration with Kepler



PARAMETERS

Job specification

Below are all job's parameters:
- inputFiles: list of local files to be uploaded
- jobType: normal, openmpi or parametric
-

You MUST check if these are correct

- 'demos/' directory location
- proxy file location
- output directory location
- RAS address

- ras: $ras_balsa
- demoLocation: E:/Kepler/
- proxyLocation: E:/grid/keys/proxy
- outputLocation: E:/Kepler/output/
- isTunnelled: true

If you need to set any specific CE/SE/WMS then set these values, otherwise leave them blank

- customSE:

Leave these unc

- predefined RAS addresses
- tables' names in the internal database
- jobs' statuses counters
- file with main jobs' ids stored

- ras_balsa: http://balsa.man.poznan.pl:808
- ras_cedrus: http://cedrus.man.poznan.pl:8
- ras_senecio: http://senecio.man.poznan.pl
- ras_stipa: http://stipa.man.poznan.pl:8080
- jobidsToCheck: to_check
- jobidsDone: done
- jobidsAborted: aborted
- jobidsFinished: finished
- doneCount: 0
- abortedCount: 0
- otherCount: 0
- finishedCount: 0
- masterJobsLocation: $demoLocation/serp
- masterJob: i-752
- jobIds: {}
- isContinueLastJob: false

- jobArguments: {"arg1=val1", "exp 1", "param3=val3"}
- jobExec: /test/euforia-test-job.sh
- jobInputFiles: {"lfn:/grid/euforia/bartek/euforia-job-monitoring.sh.bak", "lfn:/grid/euforia/bartek/euforia-job-start.sh.bak", "lfn:/grid/euforia/bartek/euforia-job-status.sh.bak", "lfn:/grid/euforia/bartek/euforia-lfc-downlo
- jobOutputDir: /grid/euforia/bartek/out

- cloud: ONE

- ONE: {cloudEndpoint="http://scchpblade09b.fzk.de:8088", imageId="ami-00000036", imageType="m1.xlarge", userAccessKey="bartekp"}
- Amazon: {cloudEndpoint="https://ec2.eu-west-1.amazonaws.com", imageId="ami-68093c1c", imageType="m1.large", userAccessKey="0B836YJ846F9C28RXH02"}

- reusePassword: true

Fig. 2. Screenshot of the Desktop offered by Kepler

The Kepler workflow orchestrator (Figure 2) provides users with an efficient and user-friendly Java based interface and workflow engine. This platform has been chosen as the main interface to project resources in EUFORIA. Within the cloud-pilot project, Kepler provides the framework for the creation of workflows that combine developed components in a coherent system. A set of Kepler actors allows the users to define and run their jobs on available cloud resources. To speed up the development process, some existing actors are re-used for user authentication, upload of input files and download of job outcome from file storage. Other actors, specific for cloud infrastructure, were implemented from scratch (this set of newly created components incorporates actors for handling job submission and job monitoring).

Kepler hides the complexity of underlying environment, so the user sees no difference to jobs running locally. To start a particular job the user just specifies application parameters and input files (that could reside on local file system). After launching the workflow, application input is uploaded automatically. The user could track a job, watching job monitoring information displayed by Kepler. When workflow ends its execution, job outcome could be found in predefined local folder on user's workstation.

4 COST EVALUATION

The aim of this section is to find out if it is financially more attractive to use a commercial cloud service or a scientific computer centre. In terms of a resource provider the question is, wether it is financially reasonable for a large scientific computer centre to provide cloud services to its users or to redirect them to commercial offerings.

For this we have defined a use case and calculate how much this would cost, when running at Amazon (AWS). We compare these prices with a theoretical model to get a good estimate for the cost and to better understand the Amazon prices.

4.1 Reference Use-Case

To keep the comparison reasonably simple, we decided to define one use-case and analyse the cost it causes. We consider a parameter scan use-case. This is a typical use-case which runs for some time and creates some output which is read by subsequent analysis runs. The use-case consists of a series of independent jobs, of which many are started. The output is stored for some time before it is retrieved and deleted. We assume the output needs to be downloaded from the cloud two times, corresponding to two analysis runs.

Please note that all these figures are purely fictional and do not correspond to an actual use-case. However, we have to define some ground truth on which a cost evaluation can be based. Table 1 shows the defined figures.

Number of jobs	1 000
Runtime per job	6 h, 1 CPU, (dual core)
Output per job	600 MB
Storage time of the output	6 months
Output retrieve count	2

Table 1. Resource requirements of our reference use case. We assume the compute jobs to be fully independent of each other.

4.2 Amazon Prices

We use official pricing of the Amazon webpage (`http://aws.amazon.com/#lastvisited12/2010`). The Amazon "Large Instance type" costs \$ 0.38 per hour and provides 2 CPU cores with 7.5 GB and 850 GB disk storage. The I/O performance is not specified, but labelled "high". External storage (Amazon S3) required for storing results costs additional \$ 0.09 per GB month and data transfer amounts to \$ 0.15 per GB. Please note that we have simplified these figures, which comprise some initial free volumes, e.g. the first GB of data transfer is for free. This summary can be found in the table below (1 USD = 0.75 EUR).

Resource	Description	Total
CPU	large instance type per hour	0.29 €
Disk	disk per GB month	0.07 €
Net	data transfer per GB	0.11 €

Table 2. Cost for computing at the Amazon compute cloud

4.3 Theoretical Cost Model

To get a better understanding on how the background of the pricing, we have built a model to understand the pricing for CPU, disk and networking. For building this model many assumptions had to be made. These assumptions are based on a fictional medium to large scale scientific computer centre. Whenever we were not sure about assumptions (e.g. the amount of storage disks to be handled by one administrator), we assumed the value leading to a higher price. Therefore accepting a higher total price for using the theoretical model, but ensuring a robust statement in case the model provides a lower price than Amazon. All figures are summarised in Table 3.

For networking we simply adopted the Amazon pricing.

For CPU we assume a price of 2 000 €per core, relating to 4 000 € for a dual-core machine, which is reasonable for a well performing large RAM configuration. We assume that one administrator can administer 1000 such machines, which is reasonable in a large datacentre with roughly 5 000 machines. Regarding electricity the estimate is based on a price of 0.12 €/kWh resulting in 1 M€ per year for electricity. The model assumes 2 M€, because the net-price KIT pays cannot be assumed in general. Furthermore, the cost for electricity comprises cost for disk networking and CPUs; we simplified the calculation by assuming the power is used by only 10 000 cores, therefore accepting a larger value for CPU cost. The same holds for the building which hosts the infrastructure. We calculate the building price only as a share of the CPUs and not the disks or network equipment. The assumption is that a building for 10 M€ can host 10 000 cores and all network and storage equipment. The building only lasts for 10 years, therefore a larger value was accepted for our CPU hours again.

The storage figure is based on the price of a disk as on the internet (1 TB for 100 €) and the assumption that one admin is required for 200 TB. This assumption is again based on the KIT data centre, where less than 10 people administer more than 8 Petabytes.

Since network cost is difficult to estimate, we use the same cost that Amazon charges.

Resource	Position	Cost	Comment	Total Cost
CPU	Price per core	2 000 €	3 Years lifetime	
	Admin per hour:	38.75 €	1 FTE 2 000 Cores/admin	
	Electricity	$2 \frac{\text{M€}}{\text{Year}}$	for 10 000 Cores includes disk	
	Real estate	10 M €	holds 10 000 Cores lasts 10 years	**12.97** $\frac{\text{ct}}{\text{CPUh}}$
Disk	Disk per GB	10 ct	1 Year lifetime	
	Admin per month:	6 200 €	1 FTE 200 TB per admin	**3.93** $\frac{\text{ct}}{\text{GBmonth}}$
Network	Data transfer	11.25 $\frac{\text{ct}}{\text{GB}}$	Same as AWS	**11.25** $\frac{\text{ct}}{\text{GB}}$

Table 3. Cost calculation of the three cost factors (CPU, Disk and Network per unit and time)

4.4 Summary of Cost

Summing up, we can see that Amazon is not really cheap. The theoretical model causes about two thirds of the Amazon costs. This gap could be attributed to the fact that we have not considered the cost for cloud software which would have to be developed before being able to provide the service. However, Eucalyptus and OpenNEbula are two viable examples for open source solutions that exist and can be used free of charge.

Resource	Amazon	Model	Factor	Total AWS	Total Model
EUR per CPU hour	0.2850	0.1297	6 000	1 710.00	778.35
EUR per GB month	0.0675	0.0393	6 000	405.00	236.00
EUR GB transfer	0.1125	0.1125	12 000	1 350.00	1 350.00
				3 465.00 €	2 364.35 €

Table 4. Cost caused by running the defined test-use case. Comparison of the actual cost that would be caused on the Amazon cloud and the virtual cost according to the cost model is shown.

We would like to stress again that for the theoretical model we chose to use higher figures, when unsure. Therefore, we can confirm that it is possible to offer cheaper cloud services to the scientific community, given a large computer centre is available.

During the research for the pricing we found one potentially interesting, but totally different figure for quantifying the value of computation. It uses the amount of generated papers per energy required to compute the results. NESRC in USA claims to have achieved 450 publications per MW year in 2009 on their HPC infrastructure. We are not aware of any comparable figure in this metric, but believe this is the ultimately correct unit.

5 BENCHMARKING PERFORMANCE

After finding that it is possible to provide cheaper cloud services than Amazon in large scientific computer centres, in this chapter we want to understand the differences in performance. In the nature of European projects this research was carried out by a different team on different resources. Therefore, this section compares the Amazon HPC to an HPC Cluster and to a Cray XT4. Please note that different pricing than that given in the previous section applies to the Amazon HPC cluster.

The virtualization used in modern clouds is generally accepted to add little in the way of overheads for computation which means that benchmarking the compute or memory performance of a standard cloud virtual machine will not provide any interesting information. However, from the point of view of scientific computational simulation it is interesting to understand the performance which can be obtained when using cloud services for computation.

5.1 Infrastructure for Benchmarks

5.1.1 Amazon EC2

Amazon's cloud infrastructure, the Elastic Compute Cloud (EC2), offers customers the opportunity to buy compute time on virtualised resources. Amazon offers a wide range of different types of compute images at different prices, from basic, low-memory images at $ 0.16 an hour, to specialized high-performance images at $ 1.60 per hour.

For the work undertaken in this study, which aims at testing the suitability of EC2 for parallel computation, the high-performance Cluster Compute images were used. Each node consists of two 2.3 GHz quad-core Intel Nehalem processors with the nodes connected using 10 Gigabit Ethernet.

5.1.2 Comparison HPC Platform: HECToR

To be able to compare the Amazon EC2 HPC performance we needed a traditional HPC system. For these benchmarks we used HECToR, the UK's national high-performance computing service. It is used for a wide range of application areas by scientists from across the UK (and Europe). The system currently consists of two different architectures (both Cray's) – for the performance tests in this report the XT4 part of the system was used. The Cray XT4 has two dual-core AMD Opteron 2.3 GHz nodes with 8 GB RAM.

5.2 Parallel Benchmark: The IMB Suite

A single Amazon virtual machine instance only offers access to 8 compute cores (all attached to the same shared memory), so in order to run parallel jobs with more than 8 processes it is necessary to start multiple instances and enable communication between them. One of these instances should act as the master, the remaining ones are treated as compute instances.

The Intel MPI Benchmarks (IMB) are a widely used set of benchmarks that are used to test some of the most important MPI features on a parallel system. They provide an overview of the performance in terms of communication bandwidth and latency. The IMB suite consists of three parts: IMB-MPI1, which addresses the classical message-passing functionalities; IMB-EXT, which focuses on single-sided communication (part of the MPI2 standard); and IMB-IO, which looks at the performance of parallel reads/writes. In this study, we concentrate on the standard message-passing performance, as well as some basic MPI-IO results.

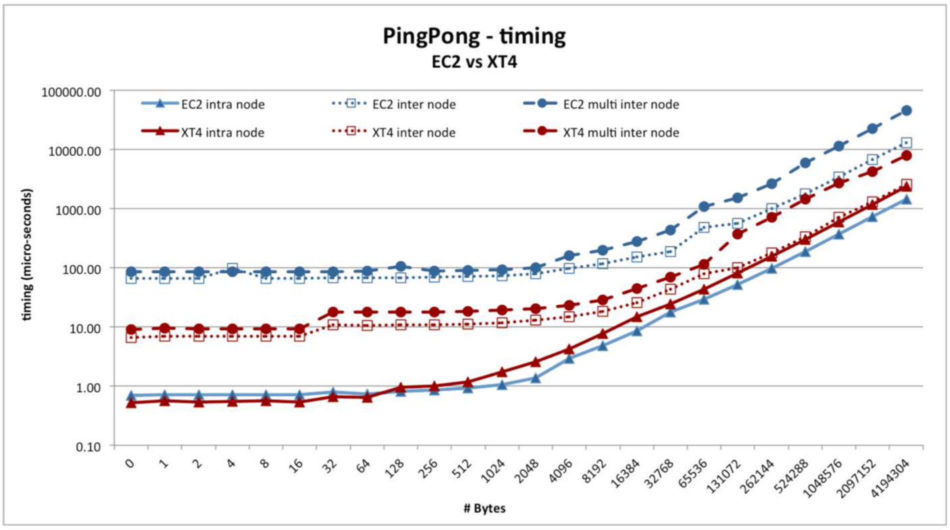


Fig. 3. Timings for the PingPong benchmark on EC2 and XT4. When running the PingPong benchmark with both processes being placed on the same node, the performance exhibited by both EC2 and XT4 is comparable. For message larger than 128 Bytes, EC2 shows better performance. This is attributed to the difference in the architecture and size of the nodes between the two systems: an XT4 node consists of a quad-core AMD Opteron processor with 8 GB of main memory, whereas an EC2 node consists of two quad-core Intel Nehalem processors with a total of 23 GB of memory.

PingPong The most basic test to measure communication overheads and throughput capabilities is the PingPong benchmark: a single message of a given size (# Bytes) is passed between two processes using MPI_Send and MPI_Receive. Figure 3 shows the timings of the PingPong benchmark on both EC2 and XT4. The graph shows three different types of runs: the intra node measurements show the performance of the benchmark between processes on the same node; for the inter node measurements, the two processes are placed on a different node each, forcing the benchmark to use the interconnect; the multi inter node test runs multiple sets of PingPong benchmarks, with each pair of processes placed on different nodes using an 8×2 mapping.

This picture changes however once the benchmark is forced to run across two nodes and use the systems' network: the XT4 is faster by an order of magnitude and the benefits of a high-performance interconnect becomes clear. The "multi" benchmarks are used to simulate a system under a full workload and all resources are used, so any artificial performance benefits from access to large amounts of main memory disappear.

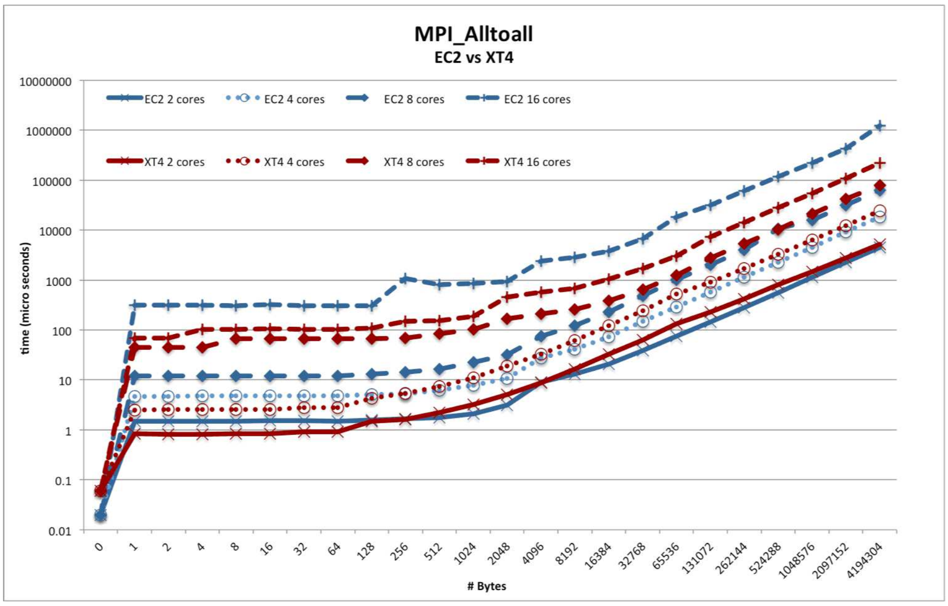


Fig. 4. Throughput levels (in Mbytes per second) for the PingPong benchmark. The graph confirms the performance picture given by the timings and shows, perhaps more clearly, how a simulated full workload affects the performance. Throughput tops out at 85 MB/s on EC2, whereas 475 MB/s are achieved on the XT4.

Alltoall While the PingPong benchmark is an example of point-to-point communication, the Alltoall benchmark tests collective communication: every process sends N bytes to all other processes and receives N bytes from those processes in return (i.e. a total of N bytes $\cdot$ # processes). The benchmark uses the MPI_Alltoall function.

On 16 cores, both systems are forced to use the interconnect – here, the superior network on the XT4 makes a significant difference (up to an order of magnitude) for all message sizes.

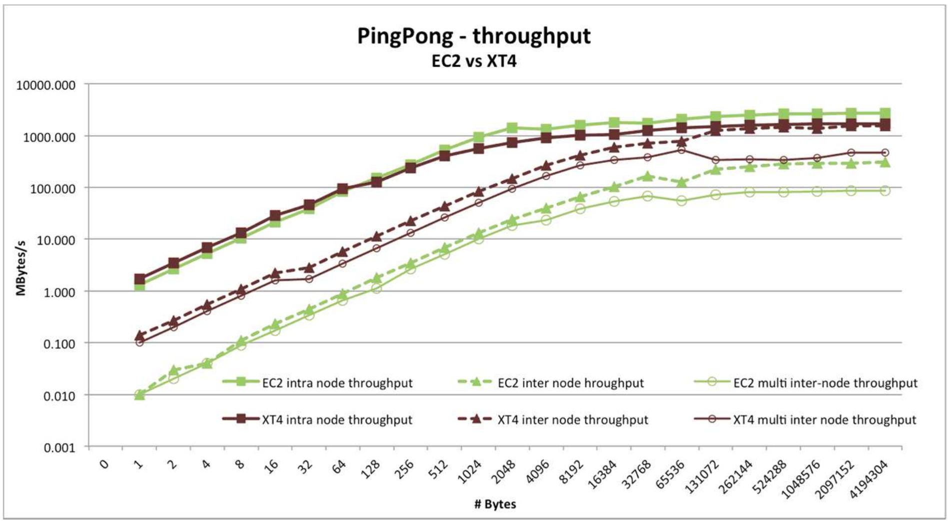


Fig. 5. Screenshot of the Desktop offered by Kepler

Fig. 6. Timings of the Alltoall benchmark from 2 to 16 cores, comparing EC2 and XT4. For 2 and 4 cores, the performance difference is relatively small, and for small message sizes, EC2 outperforms the XT4 – again, this is not surprising as the benchmarks run internal to the nodes and the more powerful Intel CPU will give EC2 the marginally better performance. For 8 cores (which fit inside a node on EC2, but across two nodes on the XT4), the performance benefit of the intra-node communication becomes even clearer for messages up to 16 KB where the timing difference is nearly an order of magnitude. However, for larger messages this benefit, which stems from the intra-node cache efficiency, mostly disappears.

Again, it is easy to spot the performance benefits of intra-node communication: for 2 and 4 cores, there is not much difference in performance between EC2 and the XT4, and on 8 cores, EC2 is slightly faster. However, the big difference again lies in the performance of those benchmarks that forced the use of the interconnect. The time to call a barrier jumps from 6 μs on 8 cores to 170 μs on 16 cores on EC2, where on the XT4 the difference is 11 s on 8 cores versus 19μs on 16 cores.

5.3 Benchmarking Conclusions

From the benchmarks we have run we can see that whilst the Amazon compute cloud does provide resources for large scale parallel programs they do not match the current performance available from existing HPC machines. The cost is comparable between Amazon and the XT4 for compute resources (Amazon is $ 1.60 per node per hour and the XT4 would be approximately $ 1.70 per node per hour). However, the cost associated with the XT4 covers all aspects of the service (compute, storage, network data transfer, etc.) whereas Amazon has extra costs for data storage and data transfers across the network.

We can also see that clouds like the NGS cloud (an academic cloud) with shared resources performance can be significantly impaired for serial applications when compared with what can be obtained using local compute servers or resources.

Another aspect which should be considered is the effort currently required by users to access and run on the cloud. The parallel nodes on Amazon required extra setup to enable large (more that 8 core) parallel jobs to be run. Furthermore, both clouds required significant work on the virtual images before they could be used for the application or low-level benchmarks. The Amazon clouds were reasonably straightforward but still required libraries to be compiled and installed on them. The NGS cloud virtual images required much more work to configure and set up the disk space and other aspects of the operating system. Once this work has been done it can be re-used but this work cannot be expected of general computational simulation scientists.

6 CONCLUSIONS

In this paper we have described and successfully implemented an architecture that enabled us to use cloud resources from within the Kepler [4] workflow environment. Both scientific and commercial cloud infrastructures can be allocated transparently using our solution.

This success leads to the question which resources should be allocated with best efficiency. We have therefore carefully calculated the cost that should be charged by a non-profit scientific computer centre and compared these with the cost charged by Amazon. This comparison indicates that a scientific computer centre can operate at two thirds of the cost that Amazon charges.

This might be attributed to a difference in performance between the two offerings. We have therefore conducted some benchmark runs which indicate that the price charged at Amazon is not justified by better performance but rather by easier usage and by the amount of available tools and features.

6.1 Future Work

Encouraged by the results of this paper, we plan to improve the cloud inclusion into Kepler by extending the solution with currently missing features. We envisage to support the deployment and develpoment cycle of sourcecode. The goal is that end users can develop on their local computer. Their output can be automatically syncronised to the cloud instances at startup.

On the infrastructure side, we plan to collaborate with EU projects to sustainably set up scientific cloud services.

REFERENCES

[1] Sotomayor, B.—Montero, R. S.—Llorente, I. M.—Foster, I.: An Open Source Solution for Virtual Infrastructure Management in Private and Hybrid Clouds. Preprint ANL/MCS-P1649-0709, 2009, http://www.mcs.anl.gov/uploads/cels/papers/P1649.pdf.

[2] Nurmi, D.—Wolski, R.—Grzegorczyk, Ch.—Obertelli, G.—Soman, S.—Youseff, L.—Zagorodnov, D.: CCGRID '09 Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid table of contents 2009, pp. 124–131, IEEE Computer Society Washington, DC, USA, 2009, ISBN 978-0-7695-3622-4, doi 10.1109/CCGRID.2009.93.

[3] Stotzka, R.—Morales-Ramos, E.—Aspnäs, M.—Aström, J.—Cárdenas-Montes, M.—Castejón, F.—Cela, J. M.–Coster, D. P.—Gómez-Iglesias, A.—Guillerminet, B.—Hammad, A.—Hardt, M.—Kos, L.—Piccioni-Koch, D.—Campos Plasencia, I.—Plociennik, M.—Poghosyan, G.—Smith, L.—Sonnendrcker, E.—Strand, P.—Westerholm, J.: EUFORIA – Simulation Environment for ITER Fusion Research. 34th Euromicro Conference, Software Engineering and Advanced Applications, Parma, Italy, September 2–5,2008, ISBN 978-3-902457-20-3. DOI: 10.1109/SEAA.2008.11, http://www.euforia-project.eu/.

[4] McPhillips, T.—Bowers, S.—Zinn, D.—Ludaescher, B.: Scientific Workflow Design for Mere Mortals. Future Generation Computer Systems, Vol. 25, 2008, pp. 541–551.

[5] Cecchi, M.—Capannini, F.—Dorigo, A.—Ghiselli, A.—Giacomini, F.—Maraschini, F.—Marzolla, M.—Monforte, S.—Pacini, F.—Petronzio, L.—Prelz, F.: The gLite Workload Management System. Advances in Grid and Pervasive Computing, LNCS Springer Berlin 2009, http://dx.doi.org/10.1007/978-3-642-01671-4_24.

Marcus Hardt received his Diploma in Physics at the RWTH in Aachen in 2001. He is the author or co-author of around 20 papers in peer-reviewed conferences and journals. He was working as founding member of WebSmart Technology GmbH and IT freelancer between 1999 and 2002. Since 2002 he has been working as a scientist at Karlsruhe Institute of Technology. In several EU funded projects (CrossGrid, int.eu.grid, Euforia) he managed the software deployment and configuration on the Europe-wide DCI and was responsible for the automated release building infrastructure. His research interest also covers ultrasound computer tomography, where he is contributing to reconstructing ultrasound signals aiming to compute 3D mammograms.