Enabling cloud interoperability with COMPSs

Fabrizio Marozzo[3], Francesc Lordan[1], Roger Rafanell[1], Daniele Lezzi[1], Domenico Talia[3],[4], and Rosa M. Badia[1],[2]

[1] Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC-CNS)
daniele.lezzi@bsc.es, roger.rafanell@bsc.es, rosa.m.badia@bsc.es,
[2] Artificial Intelligence Research Institute (IIIA), Spanish Council for Scientific Research (CSIC),
[3] DEIS, University of Calabria, Rende (CS), Italy
fmarozzo@deis.unical.it, talia@deis.unical.it,
[4] ICAR-CNR, Rende (CS), Italy

Abstract. The advent of Cloud computing has given to researchers the ability to access resources that satisfy their growing needs, which could not be satisfied by traditional computing resources such as PCs and locally managed clusters. On the other side, such ability, has opened new challenges for the execution of their computational work and the managing of massive amounts of data into resources provided by different private and public infrastructures.
COMP Superscalar (COMPSs) is a programming framework that provides a programming model and a runtime that ease the development of applications for distributed environments and their execution on a wide range of computational infrastructures. COMPSs has been recently extended in order to be interoperable with several cloud technologies like Amazon, OpenNebula, Emotive and other OCCI compliant offerings.
This paper presents the extensions of this interoperability layer to support the execution of COMPSs applications into the Windows Azure Platform. The framework has been evaluated through the porting of a data mining workflow to COMPSs and the execution on an hybrid testbed.

Keywords: Parallel programming models, Cloud computing, Data mining, PaaS

1 Introduction

The growth of cloud services and technologies has brought many advantages and opportunities to scientific communities offering users efficient and cost-effective solutions to their problems of lack of computational resources. Even though the cloud paradigm does not address all the issues related to the porting and execution of scientific applications on distributed infrastructures, it is widely recognized that, through clouds, researchers can provision compute resources on a pay-per-use basis, thus avoiding to enter in a procurement process that implies investment costs for buying hardware or access procedures to supercomputers.

Recently, several grid initiatives and distributed computing infrastructures [1] [2] [3] have started to develop cloud services in order to provide existing services through virtualized technologies for the dispatch of scientific applications. These technologies allow the deployment of hybrid computing environments where the provision of private clouds is backed up by public offerings such as Azure[4] or Amazon[5]. The VENUS-C[6] project in particular aims to support research and industry user communities to leverage cloud computing for their applications through the provision of a hybrid platform that provides commercial (Azure) and open source cloud services.

In such a hybrid landscape, there are technical challenges such as interoperability that need to be addressed from different points of view. The interoperability concept can refer to different things at many levels. It could mean the ability to keep the behaviour of an application when it runs on different environments such as a cluster, a grid or an IaaS provided infrastructure like Amazon instances. At lower level, it might refer to a single application running in many clouds being able to share information, which might require having a common set of interfaces and the ability of users to use the same management tools, server images and other software with a variety of Cloud computing providers and platforms. From a programming framework perspective these issues have to be solved also at different levels, developing the appropriate interfaces to interact with several cloud providers, ensuring that the applications are executed on different infrastructure without having to adapt them and handling data movements seamlessly amongst different cloud storages.

The COMP Superscalar[7] programming framework allows the programming of scientific applications and their execution on a wide number of distributed infrastructures. In cloud environments, COMPSs provides scaling and elasticity features allowing to adapt the number of available resources to the actual need of the execution. The availability of connectors for several providers makes possible the execution of scientific applications on hybrid clouds taking into account the above mentioned issues related to the porting of applications to a target cloud and their transparent execution with regards to the underlying infrastructure. This paper describes the developments for making COMPSs interoperable with Windows Azure Platform through the design of a specific adaptor.

The rest of the document is organized as follows. Section 2 describes the COMPSs framework. Section 3 details the developed Azure GAT Adaptor. Section 4 illustrates the porting of a data mining application to COMPSs. Section 5 evaluates the performance of the ported application. Section 6 discusses the related work. Section 7 presents the conclusions and the future work.

2 The COMPSs framework

COMPSs is a programming framework, composed of a programming model and an execution runtime which supports it, whose main objective is to ease the development of applications for distributed environments.

On the one hand, the programming model aims to keep the programmers unaware of the execution environment and parallelization details. They are only required to create a sequential application and specify which methods of the application code will be executed remotely. This selection is done by providing an annotated interface where these methods are declared with some metadata about them and their parameters.

On the other hand, the runtime is in charge of optimizing the performance of the application by exploiting its inherent concurrency. The runtime intercepts any call to a selected method creating a representative task and finding the data dependencies with all the previous ones that must be considered along the application run. The task is added to a task dependency graph as a new node and such dependencies are represented by edges of the graph. Tasks with no dependencies enter the scheduling step and are assigned to available resources. This decision is made according to a scheduling algorithm that takes into account data locality, task constraints and the workload of each node. According to this decision the input data for the scheduled task are transferred to the selected host and the task is remotely submitted. Once a task finishes, the task dependency graph is updated, possibly resulting in new dependency-free tasks that can be scheduled.

In a previous work[8], some of the authors described how COMPSs could also benefit from Infrastructure-as-a-Service (IaaS) offerings. Through the monitoring of the workload of the application, the runtime determines the excess/lack of resources and turns to cloud providers enforcing a dynamic management of the resource pool. In order to make COMPSs interoperable with different providers, a common interface is used, which implements the specific cloud provider API. Currently, there exist connectors for Amazon EC2 and for providers that implement the Open Cloud Computing Interface (OCCI)[9] and the Open Virtualization Format (OVF)[10] specifications for resource management.

The contribution presented in this paper is an extension of the COMPSs runtime to make it interoperable with the Microsoft Azure Platform-as-a-Service (PaaS) offering. COMPSs virtualizes all the resources provided by Azure as a single logical machine where multiple tasks can be executed at the same time. These extensions do not affect the programming model, keeping existing COMPSs applications unchanged and the user unaware of the technical details. Users only have to deal with the deployment of some components, as described later, on their Azure account.

3 The Azure JavaGAT Adaptor

In order to solve the interoperability issues related to the execution of tasks using an heterogeneous pool of resources in distributed environments, COMPSs adopts JavaGAT[11] as the uniform interface to underlying Grid and Cloud middlewares implemented in several adaptors. Whenever a task has to be executed on a specific resource, COMPSs manages all the data transfers and submits the task using the proper adaptor. The Azure JavaGAT Adaptor here described enriches COMPSs with data management and execution capabilities that make it interoperable with Azure and implemented using two subcomponents.

Data management is supported by a subcomponent called Azure File Adaptor. It allows to read and write data on the Azure Blob Storage (*Blobs*), to deploy the *libraries* needed to execute on Azure and to store the input and output data (*taskdata*) for the tasks. The Azure Resource Broker Adaptor, on the other side, is responsible for the task submission. Following the Azure Work Queue pattern, this subcomponent adds into a *Task Queue* the tasks that must be executed on an Azure resource by a *Worker*. The implementation of these COMPSs workers as Worker Role instances is based on a previous work on a Data Mining Cloud App framework[12]. In order to keep the runtime informed about each task execution, the status of the tasks is updated in a *Task Status Table*. The whole architecture of the Azure JavaGAT Adaptor is depicted in Figure 1.

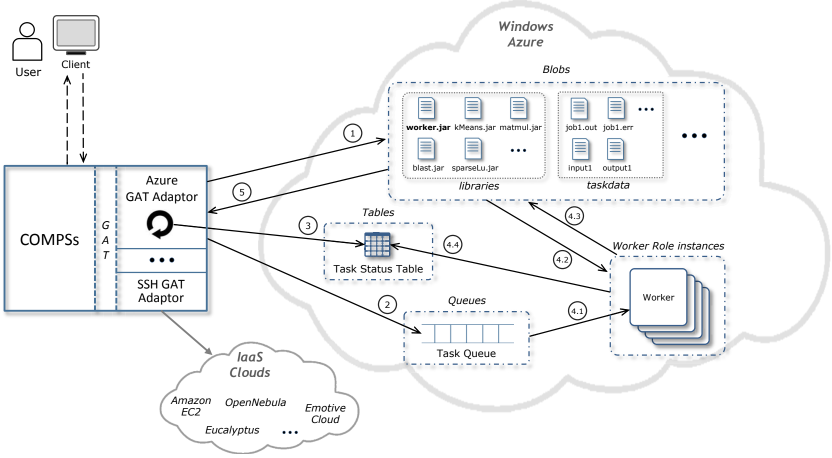


Fig. 1. The Azure GAT adaptor architecture.

The numbered components in Figure 1 correspond to each item in the list below, which describes the different stages of a remote task execution on Azure. The whole process starts when the COMPSs runtime decides to execute a dependency-free task t in the platform following the next steps:

1. The Azure GAT adaptor, through the Azure File adaptor, prepares the execution environment uploading the input application files and libraries into the Blob containers, *taskdata* and *libraries*.
2. The adaptor, via the Azure Resource Broker, inserts a task t description into the *Task Queue*.
3. The adaptor sets the status of the task t to *Submitted* in the *Task Status Table* and polls periodically in order to monitor its status until it becomes *Done* or *Failed*.

4. An idle worker W takes the task t description from the queue and, after parsing all the parameters, it runs the task. This step can be divided in the following sub-steps:
   4.1. The worker W takes the task t from the *Task Queue* starting its execution on a virtual resource. The worker sets the status of t to *Running*.
   4.2. The worker gets the needed input data and the needed libraries according to the description of t. To this end, a file transfer is performed from the Blob, where the input data is located, to the local storage of the resource, and the task is executed.
   4.3. After a task completion, the worker W moves the resulting files in the *taskdata* Blob container.
   4.4. The worker updates the status of the task in the *Task Status Table* setting it to a final status that could be *Done* or *Failed*.
5. When the adaptor detects that the task t execution has finalized, it notifies the execution end to the runtime which looks for new dependency-freed tasks to be executed. If the output files are not going to be used by any other task, the runtime downloads them from the Azure Blob.

4 Data mining on COMPSs: a classifier-based workflow

In order to validate the described work, a data mining application has been adapted to run in a cloud environment through COMPSs. Such application runs multiple instances of the same classification algorithm on a given dataset, obtaining multiple classification models, then chooses the one that classify in a more accurate way. Thus, the aim is twofold: first, validate the implementation checking that the system is able to manage the execution on different Cloud deployments; second, compare the performance of the proposed solution on an hybrid cloud scenario. The rest of the section describes the data mining application as a workflow (Section 4.1), its Java implementation (Section 4.2) and the porting to COMPSs (Section 4.3).

4.1 The application workflow

Figure 2 depicts the four general steps of the classifier-based workflow:

1. **Dataset partition:** the initial dataset is split into two parts: a training set, which trains the classifiers, and a test set to check the effectiveness of the achieved models.
2. **Classification:** during this step, the training dataset is analyzed in parallel using multiple instances of the same classifier algorithm with different parameters.
3. **Evaluation:** the quality of each classification model is measured using different performance metrics (e.g., number of misclassified items, precision and recall measure, F-measure).
4. **Model selection:** finally, the best model is selected optimizing the chosen performance metrics.

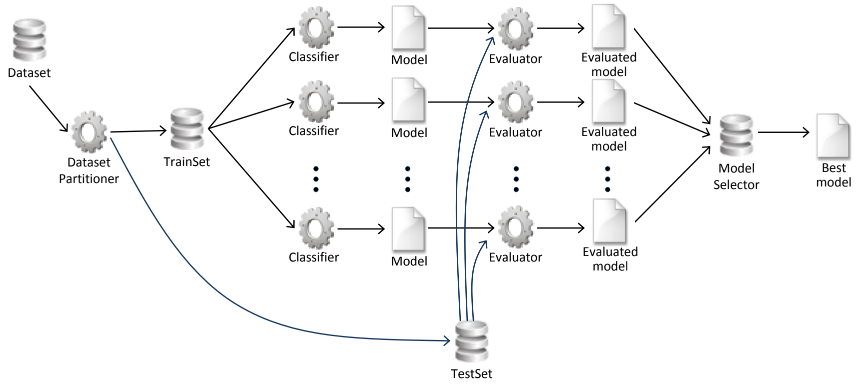


Fig. 2. The data mining application workflow

4.2 The application implementation

Following the described workflow, the initial dataset is divided in two parts: $2/3^{rd}$ are left as a training set and the remaining $1/3^{rd}$ is used as test set. The classification algorithm is the *J48*, provided in Weka[13] data mining toolkit, based on C4.5[14] algorithm. This algorithm builds a decision tree using the concept of information entropy to classify the different items in the training set. The different models are obtained varying the *confidence value* parameter of J48 in a range of values (i.e., from 0.05 to 0.50). Such range is divided in a certain number of intervals specified by the user as an application parameter. Each model is evaluated using, as a performance metric, the number of misclassified items. Listing 1.1 presents the main code of the application:

```
public static void main(String args[]) throws Exception {
  ...
  //Run remote method
  for (int i = 0; i < n_itvls; i++){
     c_val = c_min_val+i*(c_max_val-c_min_val)/(num_itvls-1);
     //***************** Remote methods ******************//
     models[i]=WorkflowImpl.learning(trainSet, c_val);
     reports[i]=WorkflowImpl.evaluate(models[i], testSet);
  }
  //Selection of the best model in binary tree way
   int n = 1;
   while (n < n_itvls){
     for (int i = 0; i < n_itvls; i+= 2 * n){
       if (i + n < n_itvls) {
         //***************** Remote method ******************//
         WorkflowImpl.getBestIndex(reports[i], reports[i+n]);
       }
     }
    n *= 2;
   }
   //Read best model
   J48 bestModel = models[reports[0].getIndex()];
}
```

Listing 1.1. Main application code.

As described in the application workflow section, the methods in lines *7* and *8* correspond to the classification and evaluation steps of the workflow. The *c_min_val* and *c_max_val* are the limits of the confidence value range, and *num_itvls* is the number of intervals specified by user. The model selection step (lines $11-22$) is performed in binary tree way in order to exploit the possibility to be parallelized by COMPSs as detailed in along the next section.

4.3 Parallelization with COMPSs: the interface

The main step of the porting of an application to COMPSs includes the preparation of a Java annotated interface provided by the programmer in order to select which methods will be executed remotely. For each annotated method, the interface specifies some information like the name of the class that implements it, and the type (e.g., primitive, object, file) and direction (e.g., in, out or in/out) of its parameters. The user can add some additional metadata to define the resource features required to execute each method. The Listing 1.2 shows the annotated interface for the presented application.

```
public interface WorkflowItf {
  @Constraints(processorCPUCount = 1, storageElemSize= 1.0)
  @Method(declaringClass = "workflow.WorkflowImpl")
  J48 learning(
     @Parameter(type = Type.OBJECT, direction = Direction.IN)
     Instances trainSet,
     @Parameter(type = Type.FLOAT, direction = Direction.IN)
     float confFactor);

  @Method(declaringClass = "workflow.WorkflowImpl")
  Report evaluate(
     @Parameter(type = Type.INT, direction = Direction.IN)
     int i,
     @Parameter(type = Type.OBJECT, direction = Direction.IN)
     J48 model,
     @Parameter(type = Type.OBJECT, direction = Direction.IN)
     Instances testSet);

   @Method(declaringClass = "workflow.WorkflowImpl")
   void getBestIndex(
     @Parameter(type = Type.OBJECT, direction = Direction.INOUT)
     Report rep0,
     @Parameter(type = Type.OBJECT, direction = Direction.IN)
     Report rep1);
}
```

Listing 1.2. Application Java interface.

The COMPSs runtime intercepts the invocations in the main code to any method contained in this interface by generating a task-dependency graph. Figure 3 shows an example of the resulting dependency graph of the data mining application. The red circles corresponds to *learning* tasks which forwards their results to the *evaluate* tasks represented in yellow, creating dependencies between them. All these evaluations end in a reduction process implemented using *getBestIndex* tasks, colored as blue, which find the model that minimizes the number of classification errors.

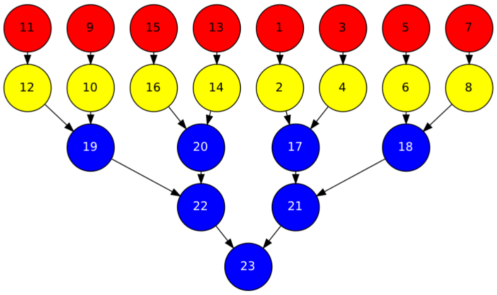


Fig. 3. Task dependency graph automatically generated by COMPSs.

5 Performance evaluation

In order to evaluate the performance of the workflow application, a set of experiments has been conducted using three different configurations: *i*) a private cloud environment managed by Emotive Cloud[15] middleware; *ii*) a public cloud testbed made of Azure instances; *iii*) an hybrid configuration using both private and public clouds.

The private cloud included a total of 96 cores available in the following way: 4 nodes with 12 Intel Xeon X5650 Six Core at 2.6GHz processors, 24GB of memory and 2TB of storage each, and 3 nodes with 16 AMD Opteron 6140 Eight Core at 2.6GHz processors, 32GB of memory and 2TB of storage each. The nodes were interconnected by a Gigabit Ethernet network and the storage was offered through a GlusterFS[16] distributed file system running in a replica configuration mode providing a total of 8TB of usable space. On this testbed 8 quad-core virtual instances with 8GB of memory and 2GB of disk space have been created running a fresh Debian Squeeze Linux distribution.

The public testbed based on Windows Azure was composed of up to 20 small virtual instances with 1.6GHz single core processor, 1.75GB of memory and 225GB of disk space each. In order to reduce the impact of data transfer on the overall execution time, the Azure's Affinity Group feature has been exploited allowing the storage and servers to be located in the same data center for performance reasons.

The *covertype*[5] dataset has been used as data source. This dataset contains information about forest cover type of a large number of sites in the United States. Each instance, corresponds to a site observation and contains 54 attributes that describe the main features of a site (e.g., elevation, aspect, slope, etc.). A subset with 290.000 instances has been taken from this dataset creating a new 36MB large one.

Table 1 presents the execution times and the speedup of an application run with 100 different models and up to 20 and 32 processors available in the public and private cloud deployments respectively. Table 2 presents the results of the

[5] http://kdd.ics.uci.edu/databases/covertype/covertype.html

same experiment running on the hybrid cloud scenario; in this case, cloud outsourcing is used to expand the computing pool out of the private cloud domain.

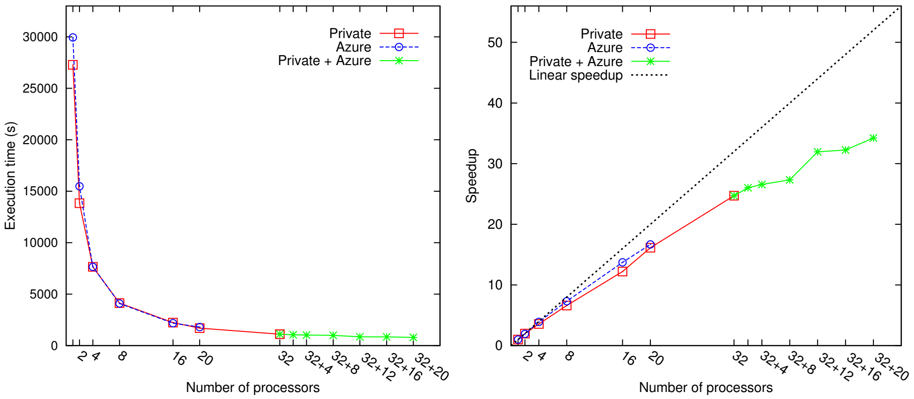
N. of cores	**Private cloud (Emotive Cloud)**		**Public cloud (Microsoft Azure)**	
	Execution time	**Speedup**	**Execution time**	**Speedup**
1	7:34:41	1	8:19:05	1
2	3:50:25	1.97	4:18:04	1.93
4	2:07:35	3.56	2:07:30	3.91
8	1:08:51	6.6	1:08:15	7.31
16	0:37:13	12.22	0:36:22	13.72
20	0:28:11	16.13	0:29:55	16.68
32	0:18:24	24.71	N/A	N/A

Table 1. Private and public cloud deployment execution times.

N. of cores	**Private cloud + Azure**	
	Execution time	**Speedup**
32 + 2	0:17:29	26.01
32 + 4	0:17:07	26.56
32 + 8	0:16:38	27.34
32 + 12	0:14:14	31.94
32 + 16	0:14:06	32.25
32 + 20	0:13:17	34.23

Table 2. Hybrid cloud deployment execution times.

As depicted in Figure 4, execution times are similar in both cases where a single cloud provider is used: Emotive Cloud and Azure. The speedup keeps a quasi-linear gain along the execution up to the point where the outsourcing starts. The trend changes observed in the speedup curve are not originated by the usage of outsourced resources but by a workload unbalance due to the impossibility to adjust the total number of tasks (constrained by the specific use case) to the amount of available resources. When the number of resources allows a good load balancing, the speedup curve recovers some of the lost performance as depicted in the 32+12 case where the gain is increased over the ideal line. Generally, when the workload does not depend on the application input, the COMPSs runtime scheduler is able to adapt the number of tasks to the number of available resources.



(a) (b)

Fig. 4. Execution time and speedup values depending on the number of processors.

6 Related work

There already exist several frameworks that enable the programming and execution of applications in the Cloud and several research products are being developed to enhance the execution of applications in the Azure Platform. MapReduce [17], a widely-offered programming model, permits the processing of vast amounts of data by dividing it into many small blocks that are processed in parallel (i.e, map phase) and their results merged (i.e., reduce phase).

Hadoop[18] is an open source software platform which implements MapReduce using the Hadoop Distributed File System (HDFS). HDFS creates multiple replicas of data blocks for reliability and places them on compute nodes so they can be processed locally. Hadoop on Azure[19] is a new Apache Hadoop based distribution for Windows Server. Microsoft Daytona[20] presents an iterative MapReduce runtime for Windows Azure designed to support a wide class of data analytics and machine learning algorithms. Also Google supports MapReduce executions in its Google App Engine[21], which provides a set of libraries to invoke external services and queue units of work (tasks) for execution. Twister[22] is an enhanced MapReduce runtime with an extended programming model that supports iterative MapReduce computations efficiently. These public cloud platforms have a high level of user's intervention in the porting of the applications requiring the use of specific APIs and the deployment and execution of the applications on their own infrastructure thus avoiding to port the code to another platform. COMPSs, on the contrary, can execute the applications on any supported cloud provider without the need to adapt the original code to the specific target platform nor writing the map and reduce functions as in MapReduce frameworks.

Manjrasoft Aneka[23] platform provides a framework for the development of application supporting not only the MapReduce programming model but also a Task Programming and Thread Programming ones. The applications can be deployed on private or public clouds such as Windows Azure, Amazon EC2, and GoGrid Cloud Service. The user has to use a specific .NET SDK for the porting of the code also to enact legacy code. Microsoft Generic Worker[24] has been extended in the context of the VENUS-C project to ease the porting of legacy code in the Azure platform. Even if the user does not have to change the core of the code, the creation of workflows is not automated, as is in COMPSs, in any of them; but has to be explicitly enacted through separated executions. Moreover, an application executed through the Generic Worker, can not be ported to other platforms.

7 Conclusions and future work

This paper presents the extensions of COMPSs programming framework to make it able to execute e-Science applications also on the Azure Platform. The contribution include the development of a JavaGAT adaptor that allows the scheduling of COMPSs tasks on Azure instances taking care of the related data transfers, and the implementation of a set of components deployed on Azure to manage the execution of the tasks internally to the instances. The proposed approach has been validated through the execution of a data mining workflow ported to COMPSs and executed on an hybrid testbed composed of a private cloud managed by Emotive Cloud and Azure machines. The results demonstrates that the runtime is able to manage and schedule the tasks on different infrastructures in a transparent way, keeping the overall performance of the application.

Future work includes the creation of a new connector in COMPSs to support the dynamic resource provisioning in Azure and enhancements to the Azure JavaGAT adaptor to optimize data transfers among different clouds, and the possibility to specify input files already available on the Azure storage. The scheduling of the COMPSs runtime will be also optimized to better balance the execution of tasks taking also into account the required time to transfer data.

Acknowledgements

This work has been supported by the Spanish Ministry of Science and Innovation (contracts TIN2007-60625, CSD2007-00050 and CAC2007-00052), by Generalitat de Catalunya (contract 2009-SGR-980), and the European Commission (VENUS-C project, Grant Agreement Number: 261565). This work was also made possible using the computing use grant provided by Microsoft in the VENUS-C project.

References

1. European Grid Infrastructure, http://www.egi.eu
2. StratusLab, http://www.stratuslab.eu

3. European Middleware Initiative, http://www.eu-emi.eu
4. Microsoft Azure, http://www.microsoft.com/azure
5. Amazon Elastic Compute Cloud, http://aws.amazon.com/es/ec2
6. Virtual multidisciplinary ENvironments USing Cloud infrastructures, http://www.venus-c.eu
7. E. Tejedor, R.M. Badia. "COMP Superscalar: Bringing GRID superscalar and GCM Together". IEEE Int. Symposium on Cluster Computing and the Grid, Lyon, France, 2008.
8. D. Lezzi, R. Rafanell, A. Carrion, I. Blanquer, R.M. Badia, V. Hernandez. "Enabling e-Science applications on the Cloud with COMPSs". Cloud Computing: Project and Initiatives, 2011.
9. Open Cloud Computing Interface Working Group, http://www.occi-wg.org
10. Distributed Management Task Force Inc., Open Virtualization Format Specification v1.1, DMT Standar DSP0243,2010.
11. G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A. Merzky, R. van Nieuwpoort, A. Reinefeld, F. Schintke, T. T. Schütt, E. Seidel, B. Ullmer. "The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid". Proceedings of the IEEE, vol. 93, no. 3, Mar 2005.
12. F. Marozzo, D. Talia, P. Trunfio. "A Cloud Framework for Parameter Sweeping Data Mining Applications". 3rd IEEE Int. Conference on Cloud Computing Technology and Science (CloudCom '11), Athens, Greece, 2011.
13. H. Witten, E. Frank. Data Mining: Practical machine learning tools with Java implementations.Morgan Kaufmann Publishers, 2000.
14. J. R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.
15. I. Goiri, J. Guitart, J. Torres. "Elastic Management of Tasks in Virtualized Environments". XX Jornadas de Paralelismo (JP 2009), Coruña, Spain, 2009.
16. GlusterFS Distributed Network File System, http://www.gluster.org
17. J. Dean, S. Ghemawat. "MapReduce: simplified data processing on large clusters". Commun. ACM, vol. 51, pp. 107-113, 2008.
18. Apache Hadoop, http://hadoop.apache.org
19. Hadoop on Azure, https://www.hadooponazure.com
20. Project Daytona, http://research.microsoft.com/en-us/projects/daytona
21. Google App Engine, http://code.google.com/intl/de/appengine
22. J. Ekanayake, Hui Li, B. Zhang, T. Gunarathne, S. Bae, J. Qiu, G. Fox. "Twister: A Runtime for Iterative MapReduce". 1st Int. Workshop on MapReduce and its Applications (MAPREDUCE'10), Chicago, USA, 2010.
23. Y. Wei, K. Sukumar, C. Vecchiola, D. Karunamoorthy, R. Buyya. "Aneka Cloud Application Platform and Its Integration with Windows Azure". CoRR, abs/1103.2590, 2011.
24. Y. Simmhan, C. Ingen, G. Subramanian, J. Li. "Bridging the Gap between Desktop and the Cloud for eScience Applications". 3rd IEEE Int. Conference on Cloud Computing (CLOUD '10), Washington, USA, 2010.