

COMPSs in the VENUS-C Platform: enabling e-Science applications on the Cloud

Daniele Lezzi^{**1,2}, Roger Rafanell¹, Francesc Lordan¹,
Enric Tejedor^{1,2}, Rosa M. Badia^{1,3}

¹ Barcelona Supercomputing Center - Centro Nacional de Supercomputación
(BSC-CNS)

² Universitat Politècnica de Catalunya (UPC)

³ Artificial Intelligence Research Institute (IIIA), Spanish Council for Scientific
Research (CSIC)

`daniele.lezzi@bsc.es`, `roger.rafanell@bsc.es`, `francesc.lordan@bsc.es`,
`enric.tejedor@bsc.es`, `rosa.m.badia@bsc.es`

Abstract. COMP Superscalar (COMPSs) is a programming framework that aims to provide an easy-to-use programming model and a runtime to enable the development of applications for distributed environments. Thanks to its modular architecture COMPSs can use a wide range of computational infrastructures providing a uniform interface for job submission and file transfer operations through adapters for different middlewares. This paper presents the developments on the COMPSs framework in the context of the VENUS-C project for the design of a programming model enactment service that allows researcher to transparently port and execute scientific applications in the Cloud.

1 Introduction

The rise of virtualized and distributed infrastructures and the emergence of multicore and GPU processing capabilities have led to a new challenge to accomplish the effective use of compute resources through the design and orchestration of distributed applications. As legacy, monolithic applications are replaced with service-oriented applications, questions arise about the key steps to be taken in architecture and design phases to maximize the utility of the infrastructures and to give to the users tools that help ease the development and design of distributed applications. Writing an application that uses the resources of a distributed environment is not as easy as writing a sequential application, and requires the programmer to deal with a number of technological concerns, like resource management or job scheduling and submission. Furthermore, programming frameworks are not currently aligned to highly scalable applications and thus do not exploit the capabilities of clouds. The design of a framework that allows the porting and execution of scientific applications on top of virtualized infrastructures, is currently a common topic in the distributed computing community. Cloud technologies are

^{**} corresponding author

mainly based on virtualisation and service orientation combined to provide elastic computing service and storage in a pay-per-use model.

One of the most common approaches to execute applications in the Cloud is MapReduce [1], a framework for data intensive distributed computing inspired by functional programming. The advantage of MapReduce is that it allows the distributed processing of the *map* and *reduce* operations. Provided that each map operation is independent of the other, all maps can be performed in parallel, though in practice it is limited by the data source and/or the number of CPUs close to those data; the other limitation is that the developer has to write the mapping and reduction functions thus forcing the researcher to explicitly adapt his application to the framework.

The Microsoft Generic Worker pattern for the Windows Azure platform supports the invocation of .NET code and Trident[2] workflows. It consists of a Worker process (a Windows Service) which runs 24x7 on a virtual machine in the Cloud. The pattern allows to run multiple machines with generic Workers concurrently, so that new work items can be distributed and scaled across a broad set of nodes. This framework is limited by the fact that generic binaries cannot be executed; instead, the .NET environment has to be used to prepare a package to be deployed in Azure compute nodes. These limitations and other improvements are being addressed in the VENUS-C project.

This paper presents the developments for extending COMPSs [3] in the context of the VENUS-C project, a recently European funded initiative whose aim is to support researchers to leverage modern Cloud computing for their existing e-Science applications. The rest of the paper is structured as follows: section 2 briefly describes COMPSs, section 3 gives an overview of the VENUS-C project with special emphasis on the job management middleware, section 4 contains the description of the effort to enable COMPSs in the VENUS-C platform and section 6 concludes the paper.

2 The COMP Superscalar Framework

COMP Superscalar (COMPSs) is a programming framework whose main objective is to ease the development of applications for distributed environments. COMPSs offers a programming model and an execution runtime.

On the one hand, the programming model aims to keep the programmer unaware of the execution environment and parallelization details. In this programming model, the programmer is only required to select the parts of the application that will be executed remotely (the so-called tasks) specifying which methods called from the application code must be executed remotely. This selection is done by simply providing an annotated interface which declares these methods, along with metadata about them and their parameters. On the other hand, the runtime of COMPSs is in charge of optimizing the performance of the application by exploiting its inherent concurrency. The runtime receives the invocation to the tasks from the application, checks the dependencies between them and decides which ones can be executed at every moment and where, considering task constraints and performance aspects. For each task, the runtime creates a node on a task

dependency graph and finds the data dependencies between the current task and all previous ones. Such dependencies are represented by edges of the graph, and must be respected when running the tasks. Tasks with no dependencies pass to the next step: the scheduling. In this phase, a task is assigned to one of the available resources. This decision is made according to a scheduling algorithm that takes into account data locality and task constraints. Next, the input data for the scheduled task are sent to the chosen host, and after that the task can be submitted for remote execution. Once a task finishes, the task dependency graph is updated, possibly resulting in newly dependency-free tasks that can be scheduled.

The applications programmed and executed with COMPSs can use a variety of Grid middlewares. Besides, some developments are currently being performed for the COMPSs framework to operate on Cloud environments as well. Such developments mainly consist in the communication of the COMPSs runtime with a Cloud scheduler, which can request the creation and deletion of virtual machines in a dynamic way. Thus, depending on the number of tasks generated by the application, COMPSs can grow and shrink the pool of virtual resources which run the tasks.

3 The VENUS-C Platform

VENUS-C develops and deploys an industrial-quality service-oriented Cloud computing platform based on virtualisation technologies, to serve research and industrial user communities by taking advantage of previous experiences and knowledge on Grids and Supercomputing. The ultimate goal is to eliminate the obstacles to the wider adoption of Cloud computing technologies by designing and developing a shared data and computing resource infrastructure that is less challenging to implement and less costly to operate.

The programming models are a major contribution of the VENUS-C project to the scientific community. In conjunction with the data access mechanisms, these programming models provide researchers with a suitable abstraction for scientific computing on top of plain virtual machines that enable them with a scientific Platform-as-a-Service.

One of the requirements of VENUS-C architecture is to define a way to support multiple programming models at the same time. In order to shield the researcher from the intricacies of the concrete implementation of different programming models, each one is enacted behind a job submission service, where researchers can submit jobs and manage their scientific workload. For this purpose, VENUS-C supports the Open Grid Forums Basic Execution Service (OGF BES) [4] and Job Submission Description Language (OGF JSDL) [5]. Each VENUS-C programming model enactment service exposes its functionality through an OGF BES/JSDL-compliant web service interface. This enactment service takes care of the concrete enactment of a job in a specific Cloud Provider.

Figure 1 depicts a high level view of the VENUS-C job management middleware architecture.

In VENUS-C, an e-Science application is separated in two parts: the core algorithmic part is ported to the Cloud through the programming models, leaving

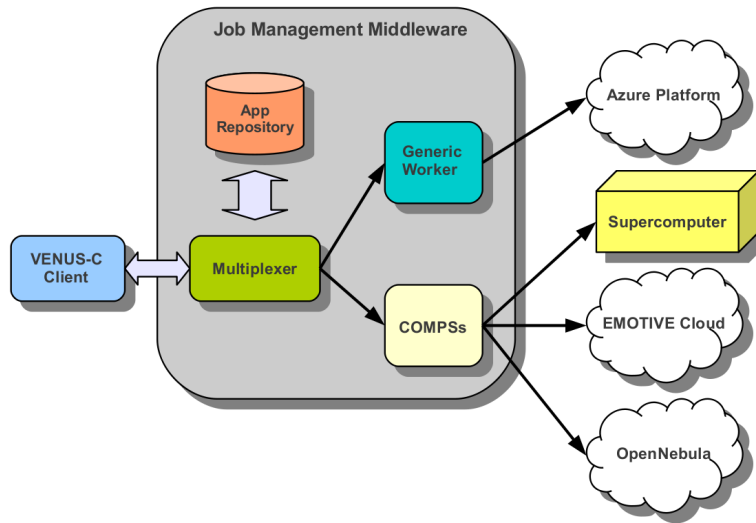


Fig. 1: The architecture of the VENUS-C Job Management Middleware

at the user the interaction with the client side part, usually a graphical user interface (GUI), to prepare and modify data, visualize results, and start the scientific computation.

The multiplexer component shown in Figure 1 acts as a unique front-end for multiple programming models, so that a client application can interact with a single OGF BES/JSDL endpoint, instead of maintaining lists and configurations of different enactment endpoints. At the current stage of the project two programming models are being enacted, COMPSs and a VENUS-C tailored implementation of the Microsoft Generic Worker [6] for the Windows Azure platform. Both COMPSs and the Generic Worker allow batch-style invocations. In the case of the Generic Worker, the command-line executable can be directly called as-is, thanks to the Generic Worker that starts a dedicated process inside the operating system. In the case of COMPSs, the application is executed using a pool of virtual machines properly allocated by the runtime or can be enqueued in a cluster through a batch system that assigns the required number of nodes.

Each enactment service enables a specific instance of an application on the underlying computational infrastructure that includes Windows Azure virtual machines and Unix virtual resources made available through several open source Cloud middlewares, specifically OpenNebula [7] and EMOTIVE Cloud [8]. Both EMOTIVE and OpenNebula exposes OCCI [9] interfaces, so that the VENUS-C programming models can programmatically manage the virtual-machine life-cycle in these Cloud infrastructures, providing the requests in the Open Virtualization Format (OVF) [10].

An application repository makes all the code packages available to the programming model enactment service, which deploys the necessary binaries, based on the specific requirements of an incoming job request.

The data has to be accessed from the Cloud-side part of the application; for this reason a Cloud SDK is made available in the project allowing the user to store the input data in the Cloud storage, to retrieve the results to the local disk and to remove data in the Cloud. The VENUS-C data management SDK supports the Cloud Data Management Interface (CDMI) [11] specification, pushed forward by the Storage Networking Industry Association (SNIA). This interface includes both a deployable web service which exposes the CDMI interface, and support libraries for different language bindings, that ease the call of the CDMI Service.

4 The VENUS-C COMPSs Enactment Service

The porting of the COMPSs framework to the VENUS-C platform includes the development of several components depicted in Figure 2. A BES compliant enactment service has been developed to allow researchers to execute COMPSs applications on the VENUS-C infrastructure. The researcher uses a client to contact the enactment service in order to submit the execution of a COMPSs application. This request is expressed through a JSDL document containing the application name, the input parameters and data references.

The same client allows the user to interact with the VENUS-C application repository to upload a packaged version of his code; such package contains the COMPSs binaries and configuration files needed by the application.

One of the advanced features of the COMPSs runtime in this platform is the ability to schedule the tasks to dynamically created virtual machines. The runtime turns to the Cloud to increase the resources pool when it detects that workload of its Workers is too high or when no Worker fulfills the constraints of a task. This increase in the number of resources is done in parallel with the submission of tasks. The requests are made using a connector to the BSC EMOTIVE Cloud middleware that has been developed implementing an OCCI client that is also used for interoperability with other providers like OpenNebula. The features of this new resource may vary depending on the need of resources that the runtime has. OVF is the chosen format to specify which constraints must accomplish the response of the Cloud Provider.

Once the request has been fulfilled and the virtual machines can be accessed, all the binaries needed by the application are deployed into it and the new resource is added to the available resources pool. Symmetrically, the runtime is able to shrink this pool by releasing some of the VMs obtained from the Cloud. This happens only when a low level of workload is detected on the Workers or when the application ends. Before releasing them, the runtime must ensure that no tasks are running in these virtual machines and there exists at least another copy of the data stored on the local disk.

One of the aims of VENUS-C is to demonstrate that existing distributed infrastructures, both data centers and supercomputing centers, can be integrated into modern Cloud environments; COMPSs helps in this vision of HPC in the Cloud, being able to dispatch workloads also into a supercomputing infrastructure processing MPI workloads. This allows traditional supercomputing infrastructures

to be provided as a service to the VENUS-C consumer.

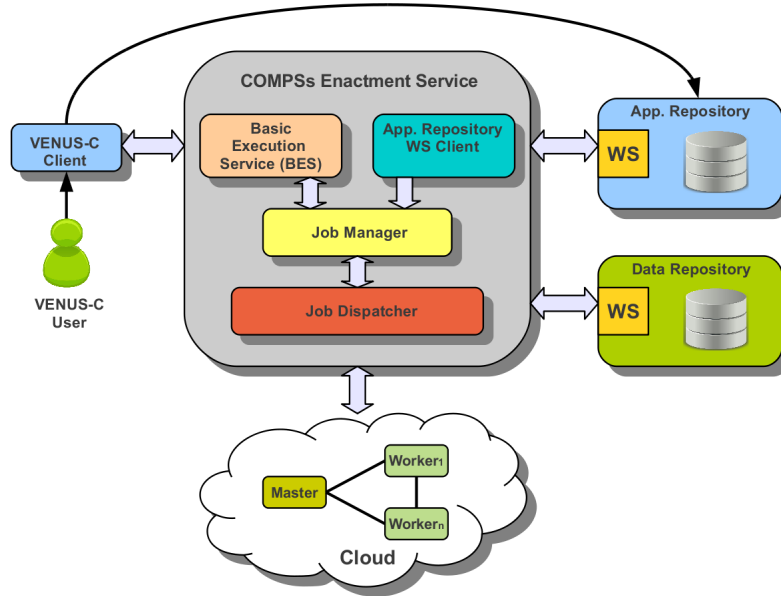


Fig. 2: The COMPSs Enactment Service architecture

4.1 The Basic Execution Service

The Basic Execution Service specification defines the service operations used to submit and manage jobs within a job management system. BES defines two WSDL port types: BES-Management for managing the BES itself, and BES-Factory for creating, monitoring and managing sets of activities. A BES manages activities, not just jobs, but the activity in the context of the enactment service is an e-Science code developed for instance as a COMPSs application. The request for an application execution is expressed in a JSDL document. The implementation of the COMPSs BES includes a Job Manager and a Job Dispatcher that actually execute the application and manage its life cycle.

The implementation of the BES-Factory port type for the COMPSs enactment services supplies the following operations:

- **CreateActivity:** This operation is used to submit a job. It takes an ActivityDocument (JSDL document) describing the application to run, and returns an identifier of a WS-Addressing EndpointReference (EPR). This EPR can be used in subsequent operations. The JSDL document is forwarded to the Job Manager that, through the *StartJob* method, creates an enactment service job

object assigning it a Universally Unique Identifier (UUID); a SAGA [12] job is created and enqueued in the Job Dispatcher which is responsible for dealing with execution as is described in the following section.

- **GetActivityStatuses:** This operation accepts a list of EPRs (previously returned from CreateActivity), and returns the state of each COMPSs job. The EPR is parsed in order to get the UUID of the job. The Job object stored in the Job Manager is requested, and its status retrieved through the *GetJob-Status* method. The state model includes a basic set of states being: *Pending*, *Running (Staging-In, Executing, Staging-Out)* and *Finished* for normal execution stages and *Failed* or *Cancelled* for abnormal execution situations. A controller for expired jobs takes care of *FAILED* or *FINISHED* jobs removing them from the list of managed jobs after a maximum allowed time using two configurable timeouts. The jobs with a *CANCELLED* end state are immediately removed from the Job Manager list.
- **GetActivityDocuments:** This operation accepts a list of EPRs and return an ActivityDocument for each EPR requested. The ActivityDocument just wraps a JSDL document containing the job description; it has been implemented for specification compatibility reasons.
- **TerminateActivities:** Takes a list of EPRs of COMPSs jobs to be terminated and returns true or false for each job depending on whether the operation was successful or not. The UUID is extracted from the EPR and the *Terminate-Job* method is called to change the status of the Job object stored in the Job Manager to *CANCELLED*; the Job Dispatcher is notified so that the job can be terminated if running or not started if still in the queue.
- **GetFactoryAttributesDocument:** This operation returns various attributes of the BES back to the client. These attributes include information about the enactment service itself, such as if it is accepting new jobs. It also contains information about the resources that the enactment service has accessed when scheduling jobs, and returns a list of Activity EPRs of the jobs controlled by the Job Manager. A BasicFilter extension can be passed to filter the results returned.

4.2 The Job Dispatcher

As seen in the previous section, the Job Manager delegates the whole control of the execution to the Job Dispatcher (see Figure 3), which maintains a user-resizable pool of threads that is responsible for picking jobs from a Job Dispatcher queue filled by the Job Manager. First a virtual machine is requested to the Cloud Provider in order to deploy the COMPSs runtime that schedule the tasks on the remote nodes. A SAGA SSH persistent connection is established with this machine where the Job Dispatcher deploys the application requesting its package to the VENUS-C application repository. This package includes the binary, the scripts for

setting the required environment variables, and other configuration files needed both by the COMPSs runtime and by the application itself.

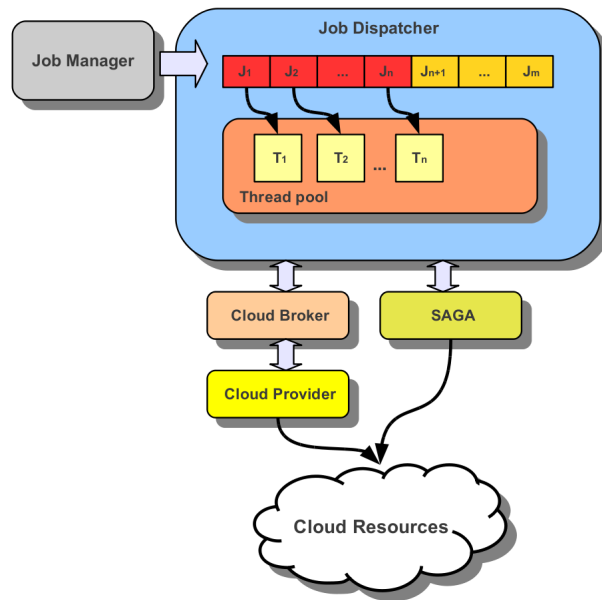


Fig. 3: The COMPSs Job Dispatcher

Once the runtime machine is deployed the COMPSs application is remotely started using the SAGA GAT adaptor. In its turn the COMPSs runtime will schedule the tasks on a set of machines created on demand. This phase includes the staging of input files from remote locations as specified in the JSDL document. The format and protocol of these location references depend on the storage provider that hosts the data; in the case of VENUS-C platform the Cloud storage is accessed in two ways, either through an SDK tailored to the Cloud storage or through a data access proxy service which can wrap multiple Cloud storage systems. In the latter case a CDMI compliant web service is made available to the clients and to the programming models. At the time of writing this paper, the CDMI service is not available and the COMPSs enactment service uses a SCP adaptor to handle the files.

When the job is completed, the output data is moved back to the storage according the user JSDL request, then the Master VM is shutdown and the information on the activity is available in the Job Manager for a time specified by the *expired jobs controller*.

5 Evaluation of an e-Science application

In order to validate the described framework a bioinformatics application has been adapted to run in a Cloud environment through the COMPSs enactment service. The aim is twofold: first, evaluating the complexity of porting the application to COMPSs in the VENUS-C platform; second, comparing the performance of the Cloud and Grid scenarios.

Hmmpfam is a widely-used bioinformatics tool included in the HMMER [15] analysis suite. It compares protein sequences with databases containing protein families, searching for significantly similar sequence matches. The work performed by hmmpfam is computationally intensive and embarrassingly parallel, which makes it a good candidate to benefit from COMPSs. A sequential Java application which wraps the hmmpfam binary was implemented in order to be run with COMPSs [16]. The main code of the application is divided in three components:

- Segmentation: the query sequences file, the database file or both are split in the main program.
- Execution: for each pair of sequence-database fragments, a task which invokes hmmpfam is run.
- Reduction: the partial outputs for each pair of sequence-database fragments are merged into a final result file by means of reduction tasks.

The input data to the workflow is the SMART database, whose size is approximately 20 MB, and a file containing 4096 protein sequences, which is 870 KB in size. The latter is partitioned in several fragments depending on the number of worker cores used in each series of tests to achieve parallelism. The output is a 320 MB file containing the scoring results.

5.1 Porting of the application

In order to port an application to COMPSs, the programmer is only required to select which methods called from the application will be executed remotely. This is done by means of an interface where the user has to specify some metadata for each method, namely: the class that implements the method (Java applications) and, for each method parameter, its type (primitive, file, ...) and direction (in, out, in/out). The user can also express constraints on the capabilities that a resource must have to run a certain method (CPU type, memory, etc).

Finally, for the Hmmpfam application to run in the VENUS-C platform, a package composed by two files was created: first, a *jar* archive containing the code to manage the three phases of the application (*Segmentation*, *Execution* and *Reduction*); second, the *hmmpfam* binary which is used in the *Execution* phase to compare each sequence fragment with the protein sequences extracted from the database. These two files are packaged in a *hmmer.tar.gz* package that the user has to upload to the application repository through the specific client. The effort required is small compared to the case of a Grid testbed, where the user has to manually specify information about the configuration of the execution testbed. In the Cloud case, the computing infrastructure is transparent to the researcher, who

only has to deal with his own code ported to COMPSs and to upload the package to the application repository.

5.2 Performances

Two tests have been run, one on a grid testbed composed by two quad-core nodes and the second one on a Cloud testbed using EMOTIVE as Cloud provider with the same number of available cores.

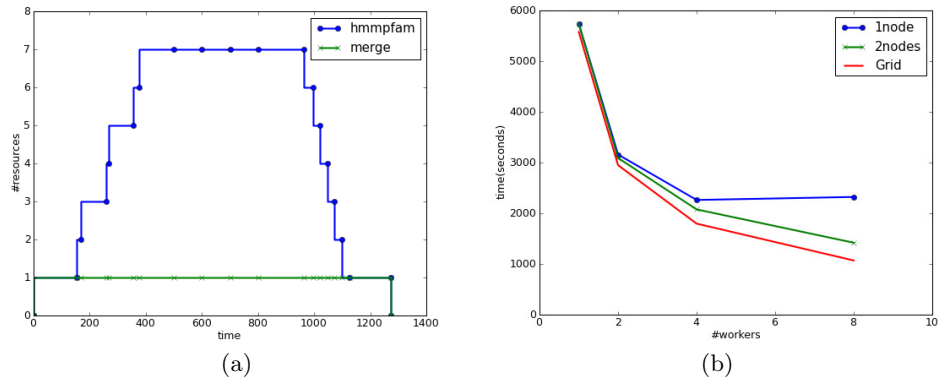


Fig. 4: Performance results.

The objective of these tests is to validate the elasticity behavior of the COMPSs runtime in the Cloud compared to a Grid testbed and the scalability of the runtime when virtual resources are used. Figure 4a depicts the evolution of the number of VMs (limited to 8) used during an Hmmpfam execution. Hmmpfam tasks are more computing intensive than the merge ones, which are shorter and data intensive. This information, provided by the user in the application interface, is used by the COMPSs runtime to ask the provider for VMs whose size fits the computing requirements of the tasks also adjusting the execution cost to the real needs of the application. COMPSs also detects the load produced by the tasks and adjust the number of required VMs able to compute them as highlighted by this example with *hmpfam* tasks. As depicted in the figure, all the *merge* tasks can be executed on the same VM since this kind of tasks does not generate enough load to need more resources.

The second test aims to measure the overhead of using cloud resources. Three configurations are used: the first one uses one physical node as cloud resource, the second one uses two nodes as cloud physical resources and the third one uses both machines as grid nodes. Figure 4b depicts the speedup of the Hmmpfam application depending on the maximum number of workers used during each execution. Due to the VM creation time the cloud execution times are always higher than the grid one using the same amount of cores. Another important aspect shown by

this test is how the number of physical nodes influences the execution time. The first limitation is the number of requests that the cloud provider used in this test can deal with at the same time; the time differences between the settings grows proportionally to the number of VMs used, fixed the number of physical machines. If the provider could deal with all the requests at the same, independently from the number of physical nodes, the differences of execution times in the grid and in the cloud would be constant. The second limitation is the number of available cores. If there are computing intensive tasks, the execution time will not decrease when the number of workers outgrows the number of cores as depicted in the single node scenario with 8 workers.

6 Conclusions and future work

This paper presented the latest efforts around the COMPSs framework in order to make it available in the VENUS-C platform. The main contribution is on the design and development of a programming model enactment service that makes the deployment and the execution of scientific applications transparent to the user. Such enactment service allows the easy porting of scientific applications to a cloud infrastructure; the effort required is minimum and consists on the registration of the COMPSs application and dependencies on a Cloud Application Repository and on the use of a client side utility to invoke the remote operations. The actual execution of the application is left to the COMPSs runtime that, using a cloud enabled adaptor for job management, schedules the tasks to virtual machines requested to different cloud providers; the requests is formatted using OVF format for the description of the resources and an OCCI adaptor to communicate with the providers.

The data management of COMPSs has been adapted to support the use of a cloud storage for input data and to store the results of the computation.

The development of a BES compliant interface for the enactment service allows on the one hand to make COMPSs usable in the context of the VENUS-C project and on the other hand to make it interoperable with other frameworks that implement the same interface.

The entire framework has been validated through the porting of an e-Science application; the results show that despite few limitations introduced by the specific Cloud infrastructure, COMPSs keeps the scalability of the application and the overall performance of its runtime while offering the researcher useful cloud features like optimized usage of resources and an easy programming and execution framework.

Future work includes the support for security to authenticate and authorize the users in order to provide compartmentalization and application security and to provide accounting information. Another improvement is the full support to the HPC Basic Profile[14] (not needed in VENUS-C platform) and the execution of interoperability tests with other implementations to validate this interface.

A specific adaptor for the Generic Worker Role will be developed in order to provide COMPSs with the capability of executing the tasks on the Azure Platform and scheduling policies will be developed in order to optimize the selection of the

resources. In the same way, scaling and elasticity mechanisms will be adopted to enhance the programming model with capabilities for scaling up or down the number of resources based on user-defined or policy driven criteria.

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), Universitat Politècnica de Catalunya (UPC Recerca predoctoral grant) and the European Commission (VENUS-C project, Grant Agreement Number: 261565)

References

1. J. Dean, S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*, Available from: <http://labs.google.com/papers/mapreduce.html>
2. The Trident framework, <http://www.microsoft.com/mscorp/tc/trident.msp>
3. E. Tejedor and R. M. Badia. COMP Superscalar: Bringing GRID superscalar and GCM Together. In 8th IEEE International Symposium on Cluster Computing and the Grid, May 2008.
4. Foster I et. al. OGSA Basic Execution Service Version 1.0. Grid Forum Document GFD-RP. 108. 8/8/2007
5. Savva A (Editor). Job Submission Description Language (JSDL) Specification, Version 1.0. Grid Forum Document GFD-R.056. 7 November 2005.
6. Simmhan Y. and Ingen v. C. (2010) Bridging the Gap between Desktop and the Cloud for eScience Applications. Available from: <http://research.microsoft.com/pubs/118329/Simmhan2010CloudSciencePlatform.pdf> Microsoft Research, U.S.
7. Open Nebula, <http://opennebula.org>
8. I. Goiri, J. Guitart, and J. Torres. (2009) Elastic Management of Tasks in Virtualized Environments.
9. Open Cloud Computing Interface Working Group, <http://www.occi-wg.org>
10. Distributed Management Task Force Inc., *Open Virtualization Format Specification v1.1*, DMTF Standard DSP0243, 2010.
11. SNIA CDMI: http://www.snia.org/tech_activities/standards/curr_standards/cdmi/
12. Tom Goodale and Shantenu Jha (2011) A Simple API for Grid Applications (SAGA) Available from: <http://www.ogf.org/documents/GFD.90.pdf>
13. EMOTIVE Cloud, <http://emotiveCloud.net>
14. The HPC Basic profile specification. <http://www.ogf.org/documents/GFD.114.pdf>
15. HMMER Analysis Suite, <http://hmmmer.janelia.org/>
16. Enric Tejedor, Rosa M. Badia, Romina Royo, Josep L. Gelpi, "Enabling HMMER for the Grid with COMP Superscalar", in proceedings of the International Conference on Computational Science 2010, ICCS2010.