Enabling e-Science Applications on the Cloud with COMPSs

Daniele Lezzi[1,2,⋆], Roger Rafanell[1], Abel Carrión[4], Ignacio Blanquer Espert[4], Vicente Hernández[4], and 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,roger.rafanell,rosa.m.badia}@bsc.es
[4] Instituto de Instrumentación para Imagen Molecular (I3M), Centro mixto CSIC, Universitat Politècnica de València, CIEMAT
iblanque@dsic.upv.es, {abcarcol,vhernand}@i3m.upv.es

Abstract. COMP Superscalar (COMPSs) is a programming framework that provides an easy-to-use programming model and a runtime to ease 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. In the context of the VENUS-C project the COMPSs framework has been extended through the development of a programming model enactment service that allows researcher to transparently port and execute scientific applications in the Cloud.

This paper presents the implementation of a bioinformatics workflow (using BLAST as core program), the porting to the COMPSs framework and its deployment on the VENUS-C platform. The proposed approach has been evaluated on a Cloud testbed using virtual machines managed by EMOTIVE Cloud and compared to a similar approach on the Azure platform and to other implementations on HPC infrastructures.

1 Introduction

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. Programming frameworks are not currently aligned to highly scalable applications and thus do not exploit the capabilities of Clouds. These technologies are mainly based on virtualization and service orientation combined to provide elastic computing service and storage in a pay-per-use model.

⋆ Corresponding author.

The first issue to be solved is the existence of multiple Cloud solutions that are not interoperable. One of the most pressing problem with respect to Cloud computing is the current difference between the individual vendor approaches, and the implicit lack of interoperability. This problem has to be solved inside the runtime of the programming framework, developing the appropriate interfaces to interact with the several Cloud middlewares thus allowing the applications to run on federated infrastructures without having to adapt the applications.

An important property of Cloud computing that poses another requirement in the design of the runtime, is infinite scaling and elasticity, i.e. the capability to provision (and de-provision) resources on demand, and to scale up or down the number of available resources as needed by the users and the application. The runtime should be able to request the provision of additional virtual resources from the underlying Cloud infrastructure. Furthermore, automated decisions could be performed based on observations of dynamic properties and system behaviors.

The COMPSs[1] framework has been recently extended in the context of the VENUS-C project, an European funded initiative whose aim is to support researchers to leverage modern Cloud computing for their existing e-Science applications. The COMPSs Enactment Service[2] provides the users of VENUS-C platform with those interoperability and transparency features with regarding to the computational infrastructure, providing dynamic scaling of resources and keeping a straightforward programming model for enabling applications in the Cloud.

This paper presents the porting of a bioinformatics application to COMPSs in the VENUS-C platform. The rest of the paper is structured as follows: section 2 briefly describes COMPSs in the VENUS-C platform, section 3 contains the description of porting a bioinformatics application to COMPSs, section 4 analyzes the performance of the ported application and section 5 concludes the paper.

2 COMPSs and the VENUS-C Platform

VENUS-C develops and deploys an industrial-quality service-oriented Cloud computing platform based on virtualization technologies, to serve to the research and industrial user communities by taking advantage of previous experiences and knowledge on Grids and Supercomputing environments. 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 resources that enable them with a scientific Platform-as-a-Service.

In order to shield the researcher from the intricacies of the concrete implementation of different programming models, each one is enacted behind a specific enactment service that researchers can use to submit jobs and manage their scientific workload. Each VENUS-C supported programming model exposes its functionality through an OGF BES/JSDL[3][4] compliant web service interface. COMPSs and the Microsoft Generic Worker[5] are the available frameworks to enable applications in the infrastructure. The Generic Worker allows the execution of binaries on the Windows Azure platform while COMPSs provides a programming framework for the definition of complex workflows.

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

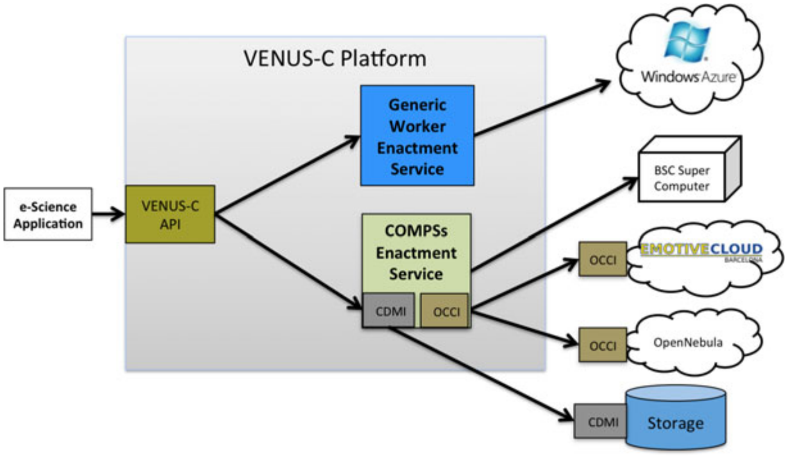

Fig. 1. The architecture of the VENUS-C Job Management Middleware

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 while the user interacts with the platform through a specific client, usually a graphical user interface (GUI), to prepare and modify data, visualize results, and start the scientific computation.

Each enactment service enables a specific instance of an application on the underlying computational infrastructure that includes Windows Azure and Unix virtual machines made available through several open source Cloud middlewares such OpenNebula[6] and EMOTIVE Cloud[7]. Interoperability with these providers is achieved through the use of an OCCI[8] connector and OVF[9] format to describe the Cloud resource capabilities. Moreover, COMPSs is also used to dispatch classical HPC workloads into a Supercomputing infrastructure allowing them to be provided as a service to the VENUS-C consumer.

The VENUS-C data management SDK supports the Cloud Data Management Interface (CDMI)[10] specification, pushed forward by the Storage Networking

Industry Association (SNIA). This interface includes both a deployable web service which exposes the CDMI interface, and the support libraries for different language bindings, that ease the call of the CDMI Service. The COMPSs enactment service implements a CDMI client that allows the access to different Cloud storage implementations through the same interface. This allows the user of the enactment service to run applications using data already available on a site thus avoiding him to be locked to a specific storage technology.

3 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 proposed solution to MPI and cloud implementations.

BLAST[11] is a widely-used bioinformatics tool for comparing primary biological sequence information, such as the amino-acid sequences of different proteins or the nucleotides of DNA sequences with sequence databases, identifying sequences that resemble the query sequence above a certain threshold. The work performed by BLAST is computationally intensive and embarassingly parallel, which makes it a good candidate to benefit from Cloud.

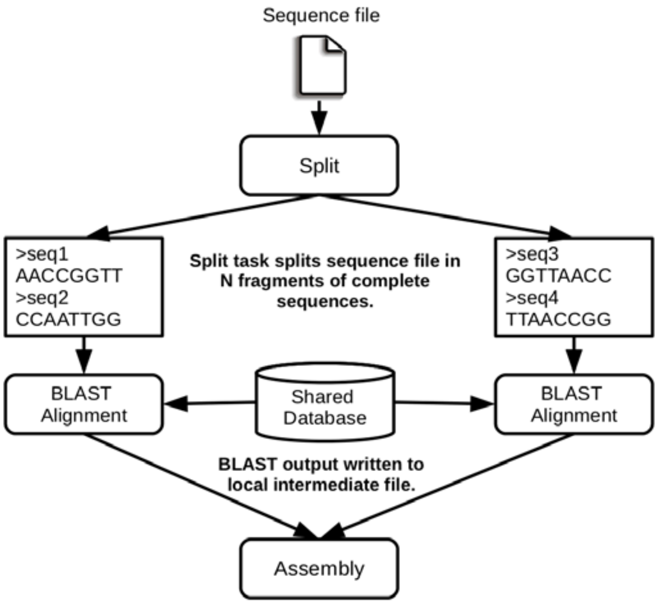

Fig. 2. The COMPSs BLAST Workflow

The BLAST workflow contains three blocks as depicted in Fig. 2:

- **Split:** the query sequences file is splitted in N fragments.
- **Alignment:** each sequence fragment is compared against the database by the blast binary.
- **Assembly:** assembly process combines all intermediate files into a single result file.

3.1 Porting of the Application

The porting of an application to COMPSs includes two steps; in the first step an interface file has to be provided by the programmer to select which methods, called from the application, will be executed remotely. The second step involves the preparation of the user code that implements these methods; in VENUS-C the interface file, the application code and the BLAST binary are assembled in a package that is stored into an application repository and deployed by the COMPSs enactment service when an execution is requested. This relieves the user of taking care of manual deployment of the binaries on the infrastructure and allows different versions of the same application to be available.

3.2 The COMPSs Application Interface

The interface declares the methods of the user application that have to be managed by the COMPSs runtime to be executed remotely; information about the method and its parameters is provided through the use of Java annotations; such metadata includes the name of the class that implements the method(s) and, for each parameter, its type (primitive, file, ...) and direction (in, out or in/out). The user can also express capabilities that a resource must fulfill to run a certain method (CPU number and type, memory, disk size, etc...).

```
public interface BlastItf {
  @Method(declaringClass = "blast.worker.BlastImpl")
  @Constraints(storageElemSize = 0.3f, processorCPUCount = 4)
  void alignment(
     @Parameter(type = Type.STRING, direction = Direction.IN)
     String db,
     @Parameter(type = Type.FILE, direction = Direction.IN)
     String fragment,
     @Parameter(type = Type.FILE, direction = Direction.OUT)
     String resFile,
     @Parameter(type = Type.STRING, direction = Direction.IN)
     String blastBinary,
     @Parameter(type = Type.STRING, direction = Direction.IN)
     String cmdArgs);
}
```

3.3 Application Implementation

In the BLAST porting, the main application code splits the input sequences file in a number of fragments specified by the user. For each fragment, the *alignment* method is called. Each generated output is assembled by the *assemblyPartitions* method.

```
public static void main(String args[]) throws Exception {
   sequences[] = split(inputFile, nFrags);
   for (fragment: sequences)
   {
     output = "resFile" + index + ".txt";
     BlastImpl.alignment(db, fragment, output, ..., cmdArgs);
     seqOutputs.add(output);
     index++;
   }
   assemblyPartitions(seqOutputs, resultFile, tempDir, nFrags);
}
```

The *alignment* method is implemented in *BlastImpl* class and simply invokes the *blastx* binary, a BLAST suite algorithm that allows six-frame conceptual translation products of a nucleotide query sequence against a protein sequence database.

```
public void alignment(String db, String fragment,
                      String resFile, ..., String cmdArgs){

   String cmd = blastBinary+ " " +"-p blastx -d " + db +
                " -i " +fragment+ " -o "+resFile+" "+cmdArgs;

   Process simProc = Runtime.getRuntime().exec(cmd);
   ...
}
```

4 Performance Analysis

In order to evaluate the described approach, a set of experiments has been conducted comparing the COMPSs BLAST implementation to another Cloud solution using the Azure Platform and to a parallel version of BLAST, mpiBLAST [12], on a parallel cluster. This tests aimed at measuring the scalability and the overall performance of the COMPSs BLAST porting using different numbers of processors and sequence fragments of the input sequence and at evaluating the Cloud overhead which involves the creation and the destruction of virtual resources on demand.

All the BLAST tests have been performed using the same use case of alignment of DNA sequences from the Sargasso Sea species against the non-redundant (NR) GenBank database with only prokaryote organisms. The input query file contains 398 sequences and the database contains 7 million of sequences with a total volume of 1 MB and 840 MB respectively.

The COMPSs porting of BLAST has been executed using the EMOTIVE Cloud middleware to manage a testbed that included two Intel Xeon Quad Core nodes at 3 GHz and 2.66 GHz respectively with 6 GB of memory, 250 GB of storage and an internal Gigabit Ethernet network. On each node a quad core VM instance has been deployed thus allowing COMPSs to execute up to 4 tasks in parallel in a single node. The input and output files are stored on a separate storage host and a network shared disk is mounted by the virtual machines. The enactment services takes care of copying data from the storage to the Cloud and moving back the result data at the end of execution.

The implementation of BLAST to the Windows Azure platform is also developed in the context of the VENUS-C project; the test used 1.6 GHz single core small Azure virtual machines instances, with 1.75 GB of memory, 225 GB of disk space and a 100 Mbps network.

The mpiBLAST version has been executed on the Tirant Supercomputer available at UPV. Tirant comprises 256 JS20 blades with IBM Power4 dual core processors at 2.0 GHz, 4 GB of memory and 36 GB of local storage. All the nodes are interconnected through Myrinet and Gigabit Network.

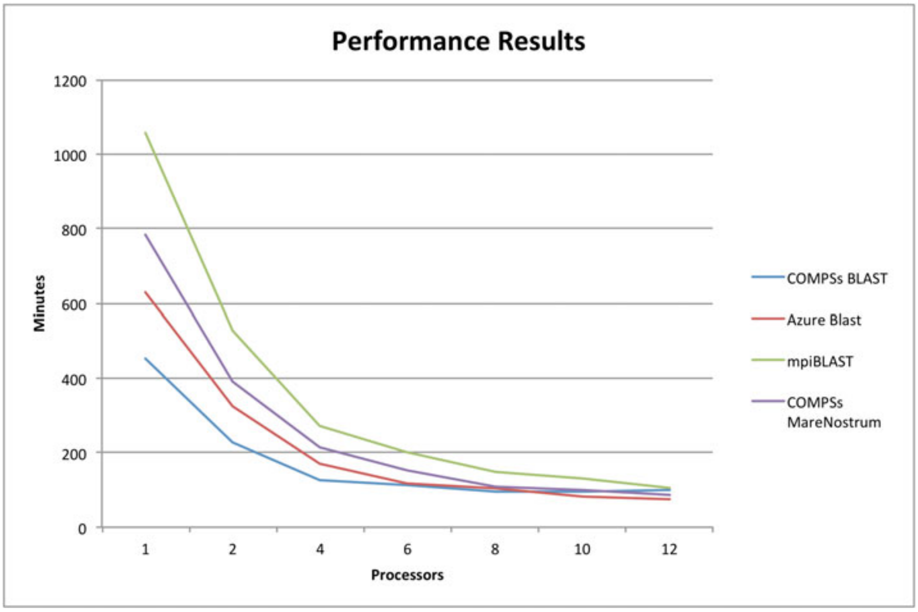

Fig. 3. Performance results

Another test has been conducted executing the COMPSs BLAST workflow on the MareNostrum Supercomputer available at BSC where the COMPSs runtime is used in production. MareNostrum comprises 2560 JS21 blades with two IBM PowerPC 970MP dual core processors at 2.3 GHz, 8 GB of memory and 36 GB of local storage. All the nodes are interconnected through Myrinet and Gigabit Network.

The results are summarized in Fig. 3. The COMPSs BLAST executed on the VENUS-C testbed using virtual resources shows better overall performance than the mpiBLAST implementation and than COMPSs BLAST executed in MareNostrum. It also performs better than the Azure execution up to 8 processors. This is due to the fact that only 2 physical nodes have been used in the tests allowing COMPSs to schedule only 8 tasks in parallel on the 8 available cores. Tests with 10 and 12 processors on the same testbed were performed deploying more virtual machines on the same host. In this case the Cloud overhead is bigger than the performance achieved through the parallel tasks scheduling. The Azure test was instead executed with an unlimited number of available resources. Nevertheless, it is worth noting that even in the worst case of limited resources, the COMPSs performance doesn't deteriorate and remains constant.

The total overhead of the COMPSs enactment service sums up to about 600 seconds on each execution; this value includes the virtual machines creation (about 200 seconds) and all the file transfers. These values can be improved using a more I/O efficient testbed than the one used for the experiments. The deployment overhead in Azure on the other side is of about 900 seconds but even removing the virtualization overhead the COMPSs BLAST implementation performs better.

5 Conclusions

This paper presented an approach for the porting and execution of scientific applications to Cloud environments through the COMPSs enactment service of the VENUS-C platform. A bioinformatics workflow based on the BLAST alignment tool has been ported to COMPSs and executed on a Cloud testbed with virtual resources managed by the EMOTIVE Cloud middleware.

The implementation of the COMPSs workflow required minimum intervention by the user who only had to provide the sequential application and an annotated interface for selecting the tasks. He interacts with the enactment service through a client that provides job and data management functionalities. It is worth noting that this implementation is not specific to the VENUS-C platform but can be used for other execution environments like clusters and grids. The same workflow for example has been also deployed on the MareNostrum supercomputer where the COMPSs framework is already offered to users to execute several scientific workflows.

The COMPSs BLAST on the VENUS-C testbed exhibited better performance than the mpiBLAST and Azure implementations considered. Even if the tests were conducted on a limited number of resources, the results are promising and 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.

A similar approach, CloudBLAST[13], implements the workflow using the MapReduce[14] paradigm to parallelize the Blast execution and a networking middleware to interconnect VMs deployed in different sites. This approach requires the user to explicitly write the map and reduce functions whereas with COMPSs there is no need to change the existing user code. Also, COMPSs runtime is able to use machines from different cloud providers without the need of deploying virtual networks thanks to the interoperability with different cloud middlewares.

Future work includes the complete interoperability of COMPSs with all the infrastructures provided in the VENUS-C. 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. Also better scheduling policies will be introduced in the COMPSs runtime 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) and the European Commission (VENUS-C project, Grant Agreement Number: 261565).

References

1. Tejedor, E., Badia, R.M.: COMP Superscalar: Bringing GRID superscalar and GCM Together. In: 8th IEEE International Symposium on Cluster Computing and the Grid (May 2008)
2. Lezzi, D., Rafanell, R., Badia, R.M., Lordan, F., Tejedor, E.: COMPSs in the VENUS-C Platform: enabling e-Science applications on the Cloud. In: Proc. of the IBERGRID 2011 Conf., Santander (June 2011)
3. Foster, I., et al.: OGSA Basic Execution Service Version 1.0. Grid Forum Document GFD-RP. 108 (August 8, 2007)
4. Savva, A. (ed.): Job Submission Description Language (JSDL) Specification, Version 1.0. Grid Forum Document GFD-R.056 (November 7, 2005)
5. Simmhan, Y., van Ingen, C.: Bridging the Gap between Desktop and the Cloud for eScience Applications, Microsoft Research, U.S. (2010), `http://research.microsoft.com/pubs/118329/Simmhan2010CloudSciencePlatform.pdf`
6. Open Nebula, `http://opennebula.org`
7. Goiri, I., Guitart, J., Torres, J.: Elastic Management of Tasks in Virtualized Environments. In: XX Jornadas de Paralelismo, JP 2009, A Corua, Spain, September 16-18, pp. 671–676 (2009)

8. Open Cloud Computing Interface Working Group, `http://www.occi-wg.org`
9. Distributed Management Task Force Inc., Open Virtualization Format Specification v1.1, DMTF Standard DSP0243 (2010)
10. SNIA CDMI, `http://www.snia.org/tech_activities/standards/curr_standards/cdmi/`
11. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic Local Alignment Search Tool. Journal of Molecular Biology 215(3), 403–410 (1990), doi:10.1006/jmbi.1990.9999
12. Darling, A., Carey, L., Feng, W.: The Design, Implementation, and Evaluation of mpiBLAST. In: Proc. of the 4th Intl. Conf. on Linux Clusters (2003)
13. Matsunaga, A., Tsugawa, M., Fortes, J.: CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications. In: IEEE Fourth International Conference on eScience (2008)
14. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: Proc. of the 6th Symp. on Operating Systems Design & Implementation, pp.137–150 (2004)