Available online at www.sciencedirect.com

ScienceDirect

SoftwareX

SoftwareX 3–4 (2015) 32–36

www.elsevier.com/locate/softx

COMP Superscalar, an interoperable programming framework

Rosa M. Badia, Javier Conejero, Carlos Diaz, Jorge Ejarque, Daniele Lezzi*, Francesc Lordan, Cristian Ramon-Cortes, Raul Sirvent

Barcelona Supercomputing Center, Spain

Received 6 May 2015; received in revised form 29 October 2015; accepted 29 October 2015

Abstract

COMPSs is a programming framework that aims to facilitate the parallelization of existing applications written in Java, C/C++ and Python scripts. For that purpose, it offers a simple programming model based on sequential development in which the user is mainly responsible for (i) identifying the functions to be executed as asynchronous parallel tasks and (ii) annotating them with annotations or standard Python decorators. A runtime system is in charge of exploiting the inherent concurrency of the code, automatically detecting and enforcing the data dependencies between tasks and spawning these tasks to the available resources, which can be nodes in a cluster, clouds or grids. In cloud environments, COMPSs provides scalability and elasticity features allowing the dynamic provision of resources.

Keywords: Parallel programming models; Interoperability; Scientific computing

Code metadata

Current code version	*1.2*
Permanent link to code/repository used of this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-15-00010
Legal Code License	*Apache 2*
Code versioning system used	*svn*
Software code languages, tools, and services used	*Java, C/C++, Python*
Compilation requirements, operating environments & dependencies	*Linux, OSX. Maven used for compilation and resolution of dependencies*
If available Link to developer documentation/manual	http://www.bsc.es/computer-sciences/grid-computing/comp-superscalar/downloads-and-documentation
Support email for questions	http://compss.bsc.es/support-compss

1. Motivation and significance

The last years have witnessed unprecedented changes in parallel and distributed infrastructures. Parallel multi-core architectures have gained widespread use; the ever-growing need of scientific applications for computing and storage capabilities has motivated the appearance of Grids; and Clouds have emerged by combining virtualisation technologies, service-orientation and business models to deliver IT resources on demand over the Internet.

The size and complexity of these new infrastructures pose a significant programming challenge. Some of these difficulties are inherent to concurrent and distributed programming, e.g. dealing with threading, messaging, data partitioning and transfer. Other issues, on the other hand, are related to the peculiarities of the particular scenario, such as the risk of vendor lock-in when writing an application for a particular Cloud provider.

* Corresponding author.
E-mail addresses: rosa.m.badia@bsc.es (R.M. Badia), francisco.conejero@bsc.es (J. Conejero), carlos.diaz@bsc.es (C. Diaz), jorge.ejarque@bsc.es (J. Ejarque), daniele.lezzi@bsc.es (D. Lezzi), francesc.lordan@bsc.es (F. Lordan), cramonco@bsc.es (C. Ramon-Cortes), raul.sirvent@bsc.es (R. Sirvent).

http://dx.doi.org/10.1016/j.softx.2015.10.004

In the face of such a challenge, programming productivity has become crucial. There is a strong need for high-productivity programming models and languages that provide a simple means for writing parallel and distributed applications, in order to improve programmer productivity on current infrastructures without sacrificing performance.

In that sense, COMPSs [1] provides a programming model and runtime system that aims to solve the above-mentioned issues, by easing application development (for, e.g. parallel applications, business or scientific workflows, and compositions of services mixing services and code) and their execution on distributed environments. The framework implements a task-based programming model that allows applications to be written following a sequential paradigm, without the need for a specific API. The COMPSs runtime detects data dependencies, and executes the code, while exploiting the parallelism that is inherent in the sequential code.

The other important feature of the COMPSs programming framework is the ability to execute the applications transparently with regards to the underlying infrastructure. A key aspect of providing an infrastructure-unaware programming model is that programs can be developed once and run on multiple backends, without having to change the implementation. This is important when portability between clouds must be achieved. In COMPSs, the programmer is freed from having to deal with the details of the specific cloud, since these details are handled transparently by the runtime. The availability of different connectors, each implementing the specific provider API (e.g Cloud providers), makes it possible to run computational loads on multiple backend environments without the need of code adaptation. In cloud environments, COMPSs provides scaling and elasticity features that allow the number of utilized resources to be dynamically adapted to the actual execution needs.

2. Software description

The COMPSs runtime has been implemented using the Java language, so the most natural programming language for new COMPSs applications is Java. Nevertheless, to simplify the porting of existing applications written in other languages, COMPSs has support also for C/C++ and Python applications.

When the sequential code is executed, the COMPSs runtime intercepts the methods invocations and replaces them with calls to the runtime that create new asynchronous tasks. Accesses to task data within the main code are also instrumented, so that the runtime can fetch the correct data values if necessary from the remote resource where the task was generated (synchronization).

Fig. 1 depicts an example of a COMPSs application, written in Java, together with the definition of an Orchestrator that includes calls to remote methods. The COMPSs runtime is in charge of creating the tasks, managing data dependencies and executing tasks using the available infrastructure. Further details about task detection, data dependency management and task scheduling can be found in [2].

Fig. 2(a) contains an example application written using the Python implementation of COMPSs, which is known as PyCOMPSs [3]. In PyCOMPSs, the tasks are identified using Python decorators, which are part of the standard Python. Fig. 2(b) shows the task dependency graph built on the fly by the COMPSs runtime.

In addition to the programming model and runtime, COMPSs provides a set of platform tools that ease: (a) implementation of COMPSs applications using an Integrated Development Environment (IDE), (b) application deployment in distributed infrastructures using the Programming Model Enactment Service (PMES) [4], and (c) execution monitoring using the Monitoring and Tracing tools. The IDE directly invokes PMES to deploy virtual appliances related to a new programmed service in the appropriate infrastructures, thus acting as a single contact point to deal with heterogeneity in the underlying platform middlewares. The COMPSs runtime provides information and usage records at execution time, so the user can follow the progress of the application. It does this through a web interface that shows real-time information on the tasks being executed, as well as indicating resource usage. When the application has finished executing, this information can be processed and visualized using tools such as Paraver [5], in order to detect bottlenecks and unbalanced parts of the application, which could be fixed to increase application performance.

3. Illustrative examples

In this section we provide two examples of PyCOMPSs applications evaluated using the MareNostrum supercomputer at BSC: DimSweep and NeuronCorr. DimSweep performs a cluster architecture exploration using the Dimemas simulator via a parameter sweep of several configuration values, including Fabric Interconnection Network latency and bandwidth, number of nodes, CPU speed, and intranode latency and bandwidth. This example is implemented in COMPSs using two different task types: one executes a Dimemas [6] simulation and the other accumulates the results. While the simulation tasks are independent, the accumulation tasks are serialized by a chain of dependencies. To avoid a long chain of these tasks at the end of the execution, these tasks are prioritized so that they are executed between simulation tasks. The experimental evaluation used a real execution tracefile and a combination of parameter configurations that generated 2,304 tasks. Fig. 3 shows the results obtained when the number of cores used for PyCOMPSs workers was varied between 16 and 512. The time on the y-axis is the total elapsed time and the speedup is computed relative to 16 workers. The results show good scaling up to 128 workers, with diminishing returns for larger processor counts.

NeuronCorr is a neuroscience data processing example that computes all mutual cross-correlations between all pairs of a set of spike data [7]. The original example was written in Parallel Python and it has been translated to PyCOMPSs. This example has two task types: one computes the cross-correlations for a block of data and the other gathers the results in a data structure. Since the gather tasks create a chain of tasks as before, they are prioritized to avoid a long serial chain at the end of the execution. The evaluation in Fig. 4 was performed using a data set that generates 2,048 tasks.

```
public interface SampleCEI {
    @Method(declaringClass = "sample.Example")
    @Constraints( processorCPUCount = 4,
                  memoryPhysicalSize = 1.0f)
    void update(
        @Parameter(direction = IN)
        int option
        @Parameter(direction = INOUT)
        Reply value
        @Parameter(type=FILE, direction = OUT)
        String log
    );

    @Service(namespace = "http://servicess.com/example",
      name = "SampleWS", port = "samplePort")
    Reply sampleService(
        @Parameter(direction = IN)
        Query query
    );

}
```

```
@Orchestration
public void SampleOE(String[] args) {

    Example e = new Example();

    Query query = new Query(args[0]);
    Reply reply = sampleOperation(query);

    e.update(3,reply,"log.txt");

    System.out.println(e.getValue());
}
```

Fig. 1. Sample application code written in Java.

```
1  result = {}
2  num_steps = 3
3  conf = init_conf()
4  for i in range(num_steps):
5      step_res = compute_step(i, conf)
6      merge(result, step_res)

7  from pycompss.api.api import compss_wait_on
8  result = compss_wait_on(result)
9  print "Result: ", result
```

(a)

```
1  class Configuration(object):
       ...

2  from pycompss.api.task import task
3  from pycompss.api.parameter import *

4  @task(returns = Configuration)
5  def init_conf():
6      return Configuration()

7  @task(returns = dict)
8  def compute_step(step, conf):
9      res = do_some_computation(step, conf)
10     return res

11 @task(dict1 = INOUT)
12 def merge(dict1, dict2):
13     dict1.update(dict2)
```

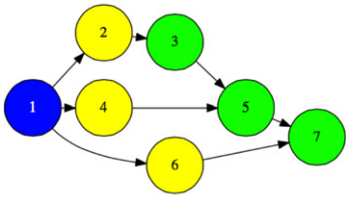

Fig. 2. Example of a sequential Python script parallelized with PyCOMPSs: (a) main program of the script, (b) task definition. On the right, the corresponding task dependency graph.

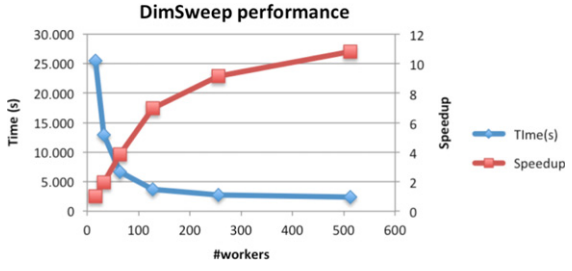

Fig. 3. Performance of DimSweep. The chart shows elapsed time and speed-up using as baseline the 16 workers case.

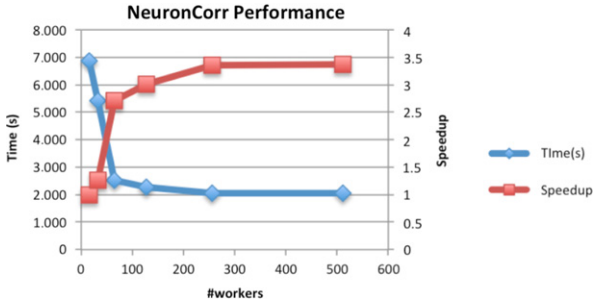

Fig. 4. Performance of NeuronCorr. The chart shows ellapsed time and speed-up using as baseline the 16 workers case.

4. Impact

The COMPSs framework is developed by the Workflows and Distributed Computing group in the Computer Sciences Department of the BSC. An important impact of COMPSs is the adoption by BSC internal users, such as Life Sciences, Earth Sciences and Computer Applications in Science and Engineering who, in turn, offer services developed using COMPSs within their communities. A clear example is the Life Sciences department, which is linked to the Spanish National Bioinformatics Institute (INB) to develop solutions for special requirements emerging from the development and execution of national research projects. COMPSs has been installed as production software on MareNostrum, for the last eight years, and is also used by the other nodes of the Spanish Supercomputing Network (RES) and by the PRACE network. There is a dedicated training programme for COMPSs in the context of PRACE Advanced Training Centres (PATCs) and COMPSs is a basic component in the activities of the BSC Severo Ochoa excellence research

program [8], where the runtime is being extended to support BSC Big Data technologies. The aim is to provide COMPSs as a high-level tool that hides the complexity of the specific framework. A relevant result is a new tool that has been developed by the BSC genomics group that constitutes the first complete and integrated solution for efficient and accurate large-scale imputation and genome-wide association analyses across multiple centers with different parallel computing environments including supercomputers and clouds [9]. COMPSs is also adopted in the Human Brain Project Flagship [10], where BSC is involved in the development of the HPC Platform, leading the activities on parallel programming models, workflows and distributed programming models, especially related to the provisioning of environments for data-intensive supercomputing.

Sustainability of the COMPSs framework is promoted through a large number of projects and collaborations with user communities. COMPSs has been adopted and extended in many projects, and has been offered as a tool for the development of scientific applications and optimization of their execution on distributed infrastructures, including in VENUS-C [11], Optimis [12] and EUBrazilOpenBio [13], and it is now leveraged in the European Grid Infrastructure (EGI) [14] as a high-level tool for porting applications to the production Federated Cloud. In EGI Federated Cloud, COMPSs represents, on the one hand, the enabling technology to transparently access the cloud infrastructure, and, on the other hand, to easily implement parallel workflows that can efficiently scale across the available resources. Examples of successful adoption of COMPSs include services offered to the biodiversity communities [15,16], implementation of pipelines for calibration, analysis and modeling for radio-astronomy data into a cloud infrastructure for the users of the LOFAR radio-telescope and the AMIGA4GAS community [17,18]. The resulting components of the latter collaboration will be part of the Square Kilometer Array (SKA) Science Data Processor (SDP) project [19]. The COMPSs group is collaborating with the BSC Life Sciences department to develop a transnational infrastructure for plant genomic science [20], in which the COMPSs framework is used to implement computational services to provide transparent access to applications and genomic data to thousands of researchers. These services are now being integrated into the technological infrastructure of the INB Spanish ELIXIR node.

In the EUBrazilCloudConnect [21] project COMPSs and PMES are used to deploy applications on the federated cloud infrastructure across Europe and Brazil. A relevant output of the adoption of COMPSs in this context is development of a new cardiovascular simulation service that allows a combination of low-definition parametric studies automatically processed in the cloud and a single high-resolution instance of the same COMPSs application in a HPC cluster. In the ASCETiC project [22] the COMPSs programming model is being extended so that it can be integrated in the framework, with identified energy efficiency parameters and metrics for Cloud services.

Between January 2015 and September 2015, the COMPSs binary packages in the BSC repository, which supports multiple Linux distributions, were downloaded more than 100 times. The same packages are also available through the EGI Marketplace, together with the customized virtual appliances that are deployed at the sites of the production infrastructure.

5. Related work

Related work includes (a) programming models for computing-intensive workloads, (b) workflow managers and (c) programming models for big data. The first group, comprising programming models for compute-intensive workloads includes frameworks for programming and execution of high-throughput applications, such as Ibis [23]. Unlike COMPSs, porting an application to Ibis requires the user has to explicitly implement an API corresponding to the specific pattern and then compile the code using specific scripts. Swift [24] is a scripting language oriented to scientific computing, which can automatically parallelize the execution of scripts and distribute tasks to various resources, exploiting implicit parallelism in a similar way to COMPSs. Unlike COMPSs, cloud deployment of a Swift application requires virtual machines to be manually provisioned before execution, and no elasticity features are provided. In the second group, comprising workflow managers, Taverna is a workflow language and computational model designed to support the automation of complex, service-based and data-intensive processes. It automatically detects tasks that are free of data-dependences, and it executes them in parallel. Deployment of cloud resources is delegated to the user, and no elasticity is provided. Pegasus [25] is another workflow manager that automatically maps high-level workflow descriptions, provided as XML files, onto distributed resources as clouds. Execution in the cloud is not straight-forward using Pegasus, because the user is forced to manually pre-deploy and configure several nodes as Condor workers. COMPSs, instead, automatically spawns the applications on dynamically-created virtual machines. In contrast to these approaches, the workflow of a COMPSs application (the dependency graph) is not defined graphically, but dynamically created as the main program runs, so that the programmer requires no knowledge of multithreading, parallel and distributed programming or service invocation. The last group of related work, comprising programming models for big data, includes models and frameworks related to the processing and generation of large data sets. The MapReduce programming model, together with frameworks based on Hadoop [26], are widely used and implemented. These frameworks provide good performance on cloud architectures, above all for data analytics applications on large data collections. Regarding the expressiveness of the models, COMPSs is more flexible than MapReduce because its applications can generate any arbitrary task graph.

6. Conclusions

COMPSs is a framework for the development, deployment and execution of parallel applications, business and scientific workflows and compositions of services, mixing services and code on distributed infrastructures. COMPSs provides users with a simple sequential programming model that does not require the use of APIs to modify the original user applications and it enables the execution of the same code on different back-

ends. The COMPSs runtime is designed to provide interoperability with different offerings through the implementation of connectors, using standards as much as possible, enabling the developed services to run on hybrid deployments.

COMPSs is part of the BSC Computer Science department's research activities on programming models for novel architectures and distributed platforms, and it is constantly being developed and extended through the analysis of the requirements of scientific communities.

The COMPSs software is open source and distributed under the Apache 2 License.

Acknowledgments

This work has been supported by the following institutions: the Spanish Government with grant SEV-2011-00067 of the Severo Ochoa Program and contract *Computacion de Altas Prestaciones VI* (TIN2012-34557); by the SGR programme (2014-SGR-1051) of the Catalan Government; by the project The Human Brain Project, funded by the European Commission under contract 604102; by the ASCETiC project funded by the European Commission under contract 610874; by the EUBrazilCloudConnect project funded by the European Commission under contract 614048; and by the Intel-BSC Exascale Lab collaboration.

References

[1] Lordan F, Tejedor EE, Ejarque J, Rafanell R, Álvarez J, Marozzo F, Lezzi D, Sirvent R, Talia D, Badia RM. ServiceSs: An interoperable programming framework for the cloud. J Grid Comput 2014;12(1):67–91.

[2] Tejedor E, Ejarque J, Lordan F, Rafanell R, Álvarez J, Lezzi D, et al. A Cloud-unaware programming model for easy development of composite services. In: Proceedings of the 3rd IEEE international conference on cloud computing technology and science, CloudCom'11. 2011.

[3] Tejedor E, Becerra Y, Alomar G, Queralt A, Badia RM, Torres J, et al. PyCOMPSs: Parallel computational workflows in Python. Int J High Perform Comput Appl 2015 [in press]. Published online before print August 19, 2015, http://dx.doi.org/10.1177/1094342015594678.

[4] Lezzi D, Rafanell R, Carrión A, Espert IB, Hernández V, Badia RM. Enabling e-science applications on the cloud with COMPSs. In: Proceedings of the 2011 international conference on parallel processing, Euro-Par'11. Heidelberg: Springer; 2012. p. 25–34.

[5] Giménez J, Labarta J, Pegenaute FX, Wen H-F, Klepacki D, Chung I-H, et al. Guided performance analysis combining profile and trace tools. In: Proceedings of the 2010 conference on parallel processing, Euro-Par 2010. Berlin, Heidelberg: Springer-Verlag; 2011. p. 513–21.

[6] Labarta J, Girona S, Pillet V, Cortes T, Gregoris L. DiP: A parallel program development environment. In: Bouge L, Fraigniaud P, Mignotte A, Robert Y, editors. Euro-Par'96 parallel processing. Lecture notes in computer science, vol. 1124. Berlin, Heidelberg: Springer; 1996. p. 665–74. http://dx.doi.org/10.1007/BFb0024763.

[7] Denker M, Wiebelt B, Fliegner D, Diesmann M, Morrison A. Practically trivial parallel data processing in a neuroscience laboratory. In: Grn S, Rotter S, editors. Analysis of parallel spike trains. Springer series in computational neuroscience, vol. 7. US: Springer; 2010. p. 413–36.

[8] Severo Ochoa Center of Excellence. http://www.bsc.es/en/severo-ochoa/home.

[9] GWImp-COMPSs: An Integrated Framework for Large-scaleGenome-wide Imputation and Association Testing. http://www.bsc.es/sites/default/files/public/u242/ds2105-boav12.pdf.

[10] Human Brain Project. https://www.humanbrainproject.eu/es/discover/the-project/overview.

[11] Lezzi D, Rafanell R, Lordan F, Tejedor E, Badia RM, COMPSs in the VENUS-C Platform: enabling e-Science applications on the Cloud. In: Proceedings of 4th iberian grid infrastructure conference. 2011. p. 73–84.

[12] Ferrer A, et al. OPTIMIS: A holistic approach to cloud service provisioning. Future Gener Comput Syst 2012;28(1):66–77. http://dx.doi.org/10.1016/j.future.2011.05.022. January 2012.

[13] Amaral R, Badia RM, Blanquer I, Braga-Neto R, Candela L, Castelli D, et al. Supporting biodiversity studies with the EUBrazilOpenBio Hybrid Data Infrastructure. Concu Comput: Practice Exp 2014; http://dx.doi.org/10.1002/cpe.3238.

[14] Lezzi D, Lordan F, Rafanell R, Badia RM, Execution of scientific workflows on federated multi-cloud infrastructures. In: Euro-Par 2013: Parallel processing workshops lecture notes in computer science, vol. 8374; 2014. p. 136–145.

[15] Lezzi D, Rafanell R, Torser E, De Giovanni R, Blanquer I, Badia RM, Programming ecological niche modeling workflows in the cloud. In: Proceedings of the 27th IEEE international conference on advanced information networking and applications (AINA-2013). 2013. http://dx.doi.org/10.1109/WAINA.2013.6.

[16] EGI Biodiversity use case. https://wiki.egi.eu/wiki/FedCloudOPENMODELLER.

[17] Sanchez Exposito S, Martin P, Ruiz JE, Verdes-Montenegro L, Garrido J, Pardell RS, et al. Web services as building blocks for science gateways in astrophysics. In: Science gateways (IWSG), 2015 7th International workshop on, vol., no., p. 80–84, 3–5 June; 2015. http://dx.doi.org/10.1109/IWSG.2015.7.

[18] EGI Radio Astronomy use case. https://wiki.egi.eu/wiki/FedCloudLOFAR.

[19] Science Data Processor in the SKA project. https://www.skatelescope.org/sdp/.

[20] transPLANT project. http://www.transplantdb.eu/project.

[21] EUBrazil Cloud Connect project. http://www.eubrazilcloudconnect.eu.

[22] K. Djemame, D. Armstrong, R. Kavanagh, A.J. Ferrer, D.G. Perez, D. Antona, et al. Energy efficiency embedded service lifecycle: Towards an energy efficient cloud computing architecture. In: CEUR Workshop Proceedings. vol. 1203; 2014. p. 1–6. CEUR Workshop Proceedings.

[23] Bal HE, Maassen J, van Nieuwpoort RV, Drost N, Kemp R, van Kessel T, et al. Real-world distributed computer with ibis. IEEE Computer 2010; 23(8):54–62.

[24] Wilde M, Hategan M, Wozniak JM, Clifford B, Katz DS, Foster I. Swift: A language for distributed parallel scripting. Parallel Comput 2011;24(9):633–52.

[25] Deelman E, Vahi K, Juve G, Rynge M, Callaghan S, Maechling PJ, et al. Pegasus, a workflow management system for science automation. Future Gener Comput Syst 2015;46C(May):17–35. http://dx.doi.org/10.1016/j.future.2014.10.008.

[26] Borthakur D. The hadoop distributed file system: Architecture and design. The Apache Software Foundation; 2007.