Providing volunteer computing at the infrastructure level to support e-science applications

Felipe Gutierrez[1], **Marcos Barreto**[1], **Antônio Tadeu Azevedo Gomes**[2]

[1]Distributed System Lab. (LaSiD), Federal University of Bahia (UFBA)
Av. Adhemar de Barros, s/n - Campus Ondina
40170-110 - Salvador, BA - Brazil

[2]National Laboratory for Scientific Computing (LNCC)
Av. Getúlio Vargas, 333 - Quitandinha
25651-075 - Petrópolis, RJ - Brazil

felipe.o.gutierrez@gmail.com, marcoseb@dcc.ufba.br, atagomes@lncc.br

Abstract. *Cloud computing has become a well-established model to deliver on demand storage and processing resources for several types of applications. It has the same characteristic of elastic processing power present in volunteer computing, which relies on an increasing number of resources shared for a limited period. In this paper, we present the integration of the BOINC volunteer computing middleware at the infrastructure level with a platform providing support for e-science applications at the user interface level. Two case studies on protein docking and number sorting are discussed in order to show how BOINC relates with such platform.*

1. Introduction

Scientific (or e-science) applications have a computationally intensive behaviour, being composed by complex tasks or large data sets that need to be executed and processed, respectively. Bioinformatics, social simulations and earth sciences are examples of such kind of applications. They demand specialized environments, such as clusters, grids, volunteer networks and clouds, that allows for task distribution, collaboration, data management, among other functionalities in order to execute.

Cluster architectures can support e-science applications by means of a significant number of processing nodes composed by multicore CPUs and, more recently, powerful GPUs. On such systems, the user can exploit the hardware heterogeneity to better allocate his application tasks aiming at high performance or reliability. Grid environments are well-established platforms to run e-science applications, as in general they are designed from scratch targeting this kind of application. Cloud platforms present a layered approach to provide hardware, databases, design environments, compilers, languages and software (applications) as "services" that can be rent by the users according to their needs. The elasticity of computing resources is one of the main goals of cloud environments, by which the number of processing nodes can be increased or reduced in order to guarantee operational requisites imposed by the application. Elasticity is also a prominent feature in volunteer computing environments, as a lot of current projects are supported by hundreds of processing nodes. BOINC[1] is arguably one of the most well-known execution environments based on volunteer computing.

The focus of this paper is on the integration of the BOINC middleware with the mc^2 (My sCientific Cloud) platform[2]. mc^2 is a Brazilian project funded by RNP (Rede Nacional de Ensino e Pesquisa) through its annual Working Groups Programme. This project aims at at implementing a platform for providing scientific applications (single tasks or workflows) with

[1]http://boinc.berkeley.edu/
[2]http://www.rnp.br/pd/gts2011-2012/GT-MCC.html

high-level user interfaces as services. The mc^2 platform offers services focused on experiment reproducibility, provenance and sharing. It relies on clusters, grids and volunteer computers at the infrastructure (execution) level, in order to better comply with different operational requisites related to memory, disk, processors, communication, performance and so on. In other words, the project harness distributed resources to perform large-scale tasks. BOINC is one of the execution environments available in the mc^2 platform. Two case studies on protein docking and number sorting are discussed in order to show how BOINC relates with such platform.

This paper is organized as follows: Section 2 presents some related work concerning the utilization of BOINC as execution environment. Section 3 presents the mc^2 project in terms of its objectives, layers and resources. Section 4 describes the approach used to integrate BOINC as an execution environment at the IaaS (Infrastructure as a Service) level of mc^2. Section 5 shows how a protein docking application can be executed on mc^2. Section 6 present another study case with a sort program. Finally, Section 7 presents some concluding remarks.

2. Related work

In this project, we decide to use BOINC as a desktop grid component, but there are other middlewares available at the internet, such as Condor [Litzkow et al. 1997] and Xtremweb [Fedak et al. 2001]. BOINC was chosen because it permits an easy interaction (i.e. a simple API) with other environments, it has a great community support and a large pool of associated projects.

There are several works related to BOINC. The EDGeS project [Farkas et al. 2010] aims at to create an integrated infrastructure that combines the advantages of service and desktop grids. It uses BOINC as a component of desktop grids and EGEE as a component of service grids. BOINC middleware is also used to expand the reach of grid computing, by combining it with others components, such as the Globus Toolkit [Myers et al. 2007]. In [Costa et al. 2011], a BOINC prototype capable of executing MapReduce jobs is presented and discussed through a simple word count application. CESC[3] is another project that relies on BOINC as execution platform for DNA sequencing and prime number calculation. A recent review on the usage of BOINC as computing platform can be found in [Korpela 2012].

The support for scientific applications running over heterogeneous platforms is also discussed in several available projects. Science Clouds[4] is a project that allows for leasing resources for short amounts of time. A user pertaining to a research community gets access to a small, medium or large set of virtual machines that can be freely used according to his application needs. Clouds and grid are also considered as execution platforms for large distributed scientific and business applications [Sripanidkulchai et al. 2010, Niehörster et al. 2009].

3. The mc^2 Platform

The purpose and main responsibilities of the mc^2 project is to offer web portals to the users that want to execute their scientific application. The way such portals are build is described elsewhere [Bastos et al. 2013] and is not the focus of this paper.

Behind such portals, a set of specialized services is provided in order to hide from the user all aspects related to resource management, communication, data transfer, etc. Figure 1 presents the mc^2 layers.

At the SaaS (Software as a Service) level, a "scientific user" is capable to execute single (executable + input data) tasks or workflows, by means of dedicated portals developed specifically for him. He does not need to know how the mc^2 platform is structured. Another kind

[3] `http://mamarreis.lsd.ufcg.edu.br/ciencia-em-sua-casa/`
[4] `http://sciencecloud.org/`

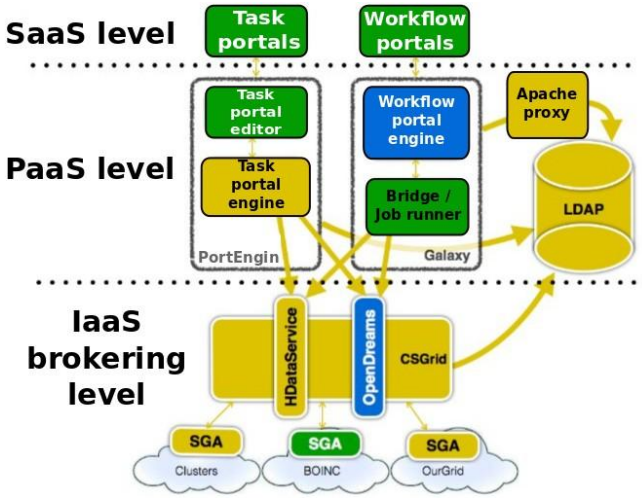

Figure 1. Layers of mc^2 platform.

of user, "the developer", works at the PaaS (Platform as a Service) level, being responsible to integrate e-science applications with PaaS tools and IaaS resources.

The PaaS level provides web tools to developers. These users are able to configure web portals for custom e-science applications. Galaxy [Goecks et al. 2010] is a framework to execute scientific workflows, focusing in reproducibility and accessibility. This tool offers great customization capacity and sharing facilities for scientific workflows. The LDAP technology [Howes and Smith. 1997] is used to integrate, authenticate and authorize almost all the components of mc^2. CSGrid is a system for grids computing that has support for management of distributed computational resources, data and users. The CSGrid and Galaxy are integrated with LDAP technology.

The IaaS level is composed by a set of distributed frameworks, namely CSGrid [Lima et al. 2005], OurGrid [Andrade et al.] and BOINC, besides specialized clusters. CSGrid is a middleware composed by a central server and several client instances (named SGA). SGA stands for "Algorithms Management System" (Sistema Gerenciador de Algoritmos from portuguese) and a BOINC SGA module was developed to integrate CSGrid and BOINC. This integration can be considered successful if a mc^2 user can operate an e-science application installed on BOINC through the CSGrid interface. CSGrid is a middleware used for grid computing, which is based on CSBase [Lima et al. 2006], a framework for resource management and implementation of algorithms in distributed and heterogeneous computing environments.

CSGrid provides a service control, user access to hard disk areas, algorithms and machines. It facilitates the visualization of the file system, its permissions to read and write and send event notifications. The SGA is a daemon tool installed on each machine, that monitors the state of this machine and launchs processes. The integration of CSGgrid and SGA is implemented in Lua and C++, and supported through CORBA in a SSI layer. When the SGA is activated, it assigns to SSI and became available to the clients already connected to the SSI. After that, a client can request the execution of some algorithm through SSI and monitor the process until its conclusion. If the connection with SSI is interrupted, the SGA remains active. When the connection is re-established, the SGA reassigns to the SSI again, stabilizing communication with the CSGrid. Figure 2 shows this integration. For laboratory tests this project was implemented using NFS (Network File Systems) to the comunication between the CSGrid and SGA, but for wide area networks the project uses CSFS, a module from CSGrid to share files on network.

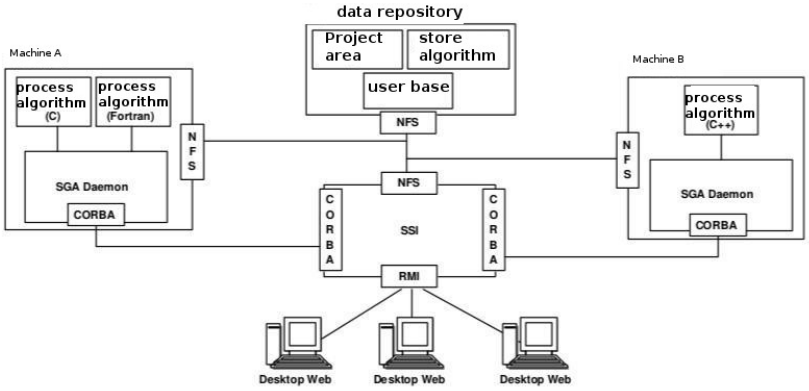

Figure 2. CSGrid infrastructure [Lima et al. 2005]

The motivation to use the CSGrid framework is the increasing need for the coordinated use of resources in distributed, heterogeneous computing environments [Foster et al. 2008]. These types of systems have a variety of platforms, applications and data decentralization. The architecture of the system considerably increases the computational power to end users when these computing resources are geographically distributed.

4. The SGA (Algorithms Management System) for BOINC

BOINC is a software platform that involves volunteer computing and grid computing. It was projected to support applications that have large computing requirements. The main requirement of the application is that it can be divided in a large number of jobs that can be done independently. So, it is better to use the BOINC platform if a lot of cheap computing processes are necessary. To implement this, the BOINC platform can be set as volunteer computing or grid computing.

Within the CSGrid infrastructure, the SGA is responsible for the execution of algorithms (applications) on behalf of the CSGrid user. In our case, CSGrid evaluates the execution flow and interacts with the SGA at the BOINC server in order to request the execution of a given application in one or more BOINC client machines. It is possible to add more BOINC client machines as needed. Figure 3 shows the interaction among all these components.

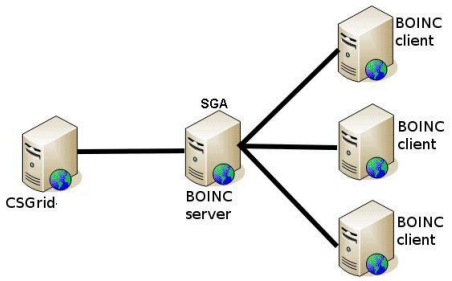

Figure 3. Communication among SGA BOINC components.

It is necessary to configure one file on the SGA machine in order for this machine to be recognized as a computing node by the CSGrid server. This file is named sgad-cnf.lua and its main part is presented in Figure 4. The attribute name is the localhost name of the SGA node. The two other attributes that are mapped to directories are the local where the project of CSGrid is build on SGA and where the algorithm is going to run. The last attribute is the name of the lib file (SGA-BOINC.lua). This file is used to create tasks on BOINC server.

Figure 4. The sgad-cnf.lua file.

The SGA program is installed at the BOINC server machine by a developer user. Through the SGA interface it is possible to set how many tasks the user wants to create and the input files to process. It is also possible to check the state of the tasks at the BOINC client machines. Figure 5 shows the SGA interface, which is accessible through the CSGrid application.

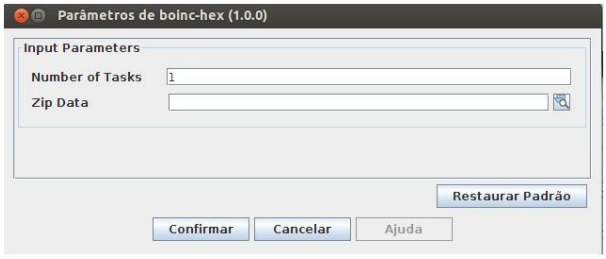

Figure 5. SGA BOINC interface.

5. Case study on Protein Docking

Hex [Ritchie 2003] is an application for interactive protein docking and molecular superposition. This program recognizes structures of proteins and DNA in PDB format, as well as read SDF file that describes small molecules.

The Hex program can be found at the INRIA portal[5]. The application uses a hex.bat file to read four input files. This file is presented in Figure 6. The file hex6i.x64 is the core of the program. The application reads four input files and generates four output files. Figure 7 presents the two more important input files to process a Hex application. Hex has three output files and one log file. They are very similar to the last input file, showing the values processed from the last two input files.

```
DISC_CACHE 0
SESSION_LOG 1
RUN_MACRO 1CLV_r_u-1CLV_l_u.mac
```

Figure 6. File hex.bat.

However, to integrate the Hex program with BOINC, it is necessary to concentrate all inputs into three files. The first one is the executable script (hex.bat). The second file is a zip file containing the Hex application and related libraries (hex6i-linux64.zip). The last file is a zip file with the input data (inputExemplo.zip). The BOINC server is able to decompress all these files and execute the Hex application, sending the tasks to be processed at the BOINC clients. The server does it through a wrapper function[6]. There are wrappers for different operating systems. In this project, we used a wrapper for Linux x64 bits systems.

[5] http://hex.loria.fr/dist68/

[6] http://boinc.berkeley.edu/trac/wiki/WrapperApp/

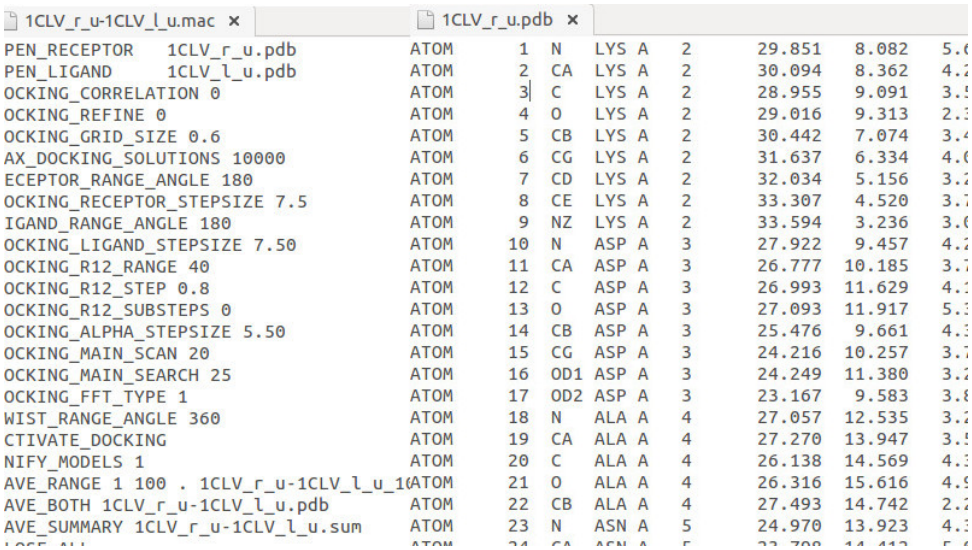

Figure 7. Hex input files.

Once the application is deployed at the BOINC server, it is possible to create tasks to process the input files and attach the necessary zip data input file. It is possible to check the state of the tasks on the BOINC-Hex web portal, a web service provided by BOINC server, and also to download the results when they are complete[7]. When the BOINC client starts to process the tasks, all application is downloaded from the BOINC server machine. The BOINC server knows how to use all core processors available at client machines. It allocates tasks according to the number of available processors. When the process ends, the results are uploaded to the server and the user can access the web site to download the output files.

It is possible to check the time spent to process the job in the hex_job_X_0_2 log file, where X is the number of the job. This file is the new name of the output file job.log, but there is a job.log for all tasks. The web portal's address is http://BOINC-IP/hex_ops, where IP relates to the BOINC Server machine and the prefix hex is the name of the application deployed on BOINC server. Also through this link, it is possible to see the task name and find the respective directory.

The mc^2 execution flow starts at CSGrid, passes through the BOINC server and reachs the BOINC clients. When the process finishes, the files come back to BOINC server and the user can download them through CSGrid. It is also possible to download the files from the BOINC-Hex web portal, as said before. But to improve the access of the scientific users, this function was developed to integrate BOINC server with CSGrid. So the entire access flow goes through the CSGrid. The process through BOINC server to the BOINC client and back to BOINC server is autonomic. So this is a good point to compare a Hex process running alone and a Hex process running under BOINC. Table 1 shows the time in minutes to process the Hex application by itself and after running on BOINC client. We use the same machine on both analyses, an Intel Core 2 Duo CPU E4400 2.0GHz x64 bits processor and 1GB of RAM. The time exceeded on the BOINC was for the download and upload the Hex application and its input and output files to the BOINC machines.

6. Case study on number sorting

We developed another application just as a proof-of-concept for SGA BOINC. A sorting program based on bubblesort that reads a file with numbers and write these numbers ordered in

[7] `http://boinc.berkeley.edu/trac/wiki/RemoteOutputFiles`

Table 1. Time to process HEX application.

Application	Time in minutes
Hex	19.23
BOINC with Hex	22.01

another file. To deploy a C++ program in BOINC[8] it is necessary to include some libraries and BOINC functions. The program is larger than the original and apparently because of that its processing will be slower. Thinking about this some measurements were made only with the program with the BOINC libs, been executed alone and into the BOINC platform.

The table 2 shows the results for this program running standalone and at the BOINC platform with different amounts of numbers. It can be noticed that the difference in processing time on both execution settings remains stable as the amounts of numbers for ordering increases. Therefore, the processing is not affected when deploying the program in the BOINC platform.

Table 2. Time to process Bubblesort program.

Quantity	Standalone (min)	BOINC (min)
100,000	1.4393	2.1929
250,000	5.2126	6.3829
500,000	20.4487	21.5930
750,000	46.3036	47.1282
1,000,000	82.4396	84.0247

To measure the execution time, the same function that creates tasks on BOINC server creates a file where the output files will be upload to the BOINC clients. When the clients upload the results back to the server, it is possible to compare the times they were created. We made both measures in the same machine we installed the BOINC client application. This machine has an Intel Core 2 Duo CPU E4400 2.0GHz x64 bits processor and 1GB of RAM.

Since the objective of the mc^2 project is to allow distributed process, the next experiment considered two machines, with the same hardware and operating system, connected to the BOINC server. These machines have two cores each, so they can start two process each one. Using the same method, we compare the time while adding files to process. When we upload four files, each machine got two files to process on their two cores. When we add more files it is possible to check that the time grows less than the number of files. Thus, it is possible to verify that decreases the processing time of the mc^2 projects while increases the number of machines BOINC client and the number of tasks to process. The table 3 shows the results in minutes for different quantity of files.

Table 3. Time to process Bubblesort program in parallel execution

Quantity	4 files	8 files	12 files	16 files	20 files	24 files
100,000	2.3909	3.4605	4.0975	5.5348	6.2184	8.2448
250,000	6.2075	11.2920	17.1688	23.3935	29.2433	34.5128

7. Concluding remarks

Through the measurements, it is possible to conclude that the extra time (almost 3 seconds in Hex application) is related to data movement between the BOINC server and clients, to

[8] `http://boinc.berkeley.edu/trac/wiki/BasicApi`

send files to be processed and to gather back the results when available. On the bubblesort C++ program, the time to transfer the files is the same even when the numbers to process was increasing. Therefore, it is possible to conclude that as the load rises, it is better to deploy the program at the mc^2 platform. With these results, it is possible to release the IaaS layer of the mc^2 system to integrate with PaaS layer. It is also possible to share the data processed on BOINC with data processed on OurGrid and other clusters on the IaaS layer.

References

Andrade, N., Cirne, W., Brasileiro, F., and Roisenberg, P. OurGrid: An approach to easily assemble grids with equitable resource sharing.

Bastos, B. F., Moreira, V. M., and Gomes, A. T. A. (2013). Rapid prototyping of science gateways in the brazilian national HPC network. In *Proceedings of the 2013 International Workshop on Science Gateways (IWSG)*. (*Accepted for publication*).

Costa, F., Silva, L., and Dahlin, M. (2011). Volunteer cloud computing: MapReduce over the Internet. In *Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 1855–1862.

Farkas, Z., Kacsuk, P., Balaton, Z., and Gombás, G. (2010). Interoperability of BOINC and EGEE. volume 26, pages 1092–1103. Elsevier, future generation computer systems edition.

Fedak, G., Germain, C., Neri, V., and Cappello, F. (2001). XtremWeb: A generic global computing system. Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID 01).

Foster, I., Zhao, Y., Raicu, I., and Lu, S. (2008). Cloud computing and grid computing 360-degree compared. pages 1–10. Grid Computing Environments Workshop, 2008. GCE 2008.

Goecks, J., Nekrutenko, A., and Taylor, J. (2010). Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol, 11(8):r86 edition.

Howes, T. and Smith., M. (1997). LDAP: programming directory-enabled applications with lightweight directory access protocol. Sams Publishing.

Korpela, E. J. (2012). SETI@home, BOINC, and volunteer distributed computing. *Annual Review of Earth and Planetary Sciences*, 40(1):69–87.

Lima, M. J. D., Melcop, T., Cerqueira, R., Cassino, C., Silvestre, B., Nery, M., and Ururahy, C. (2005). CSGrid: um sistema para integração de aplicações em grades computacionais. Anais do 23o. Simpósio Brasileiro de Redes de Computadores - SBC.

Lima, M. J. D., Ururahy, C., Moura, A., Melcop, T., Cassino, C., Nery, M., Silvestre, B., Reis, V., and Cerqueira, R. (2006). CSBase: A framework for building customized grid environments. Third International Workshop on Emerging Technologies for Next-generation GRID.

Litzkow, J. B. M., Tannenbaum, T., and Livny, M. (1997). Checkpoint and migration of unix processes in the condor distributed processing system. University of Wisconsin.

Myers, D. S., Bazinet, A. L., and Cummings, M. P. (2007). Grid computing for bioinformatics and computational biology. pages 71–85. 2 edition.

Niehörster, O., Birkenheuer, G., Brinkmann, A., Blunk, D., Elsässer, B., Herres-Pawlis, S., Krüger, J., Niehörster, J., Packschies, L., and Fels, G. (2009). Providing scientific software

as a service in consideration of service level agreements. In *Proceedings of the Cracow Grid Workshop (CGW)*, pages 55–63.

Ritchie, D. (2003). Evaluation of protein docking predictions using hex 3.1 in capri rounds 1 and 2. volume 52, pages 98–106.

Sripanidkulchai, K., Sahu, S., Ruan, Y., Shaikh, A., and Dorai, C. (2010). Are clouds ready for large distributed applications? *SIGOPS Oper. Syst. Rev.*, 44(2):18–23.