A Genetic Programming infrastructure profiting from public computation resources

F. Chávez de la O.
Centro Universitario de Mérida
C/ Sta Teresa de Jornet 38
06800 mérida (spain)
fchavez@unex.es

Manuel Rubio del Solar
CETA-CIEMAT
Paseo Ruiz de Mendoza 8
10200 Trujillo (Spain)
manuel.rubio@ciemat.es

J.L. Guisado
Centro Universitario de Mérida
C/ Sta Teresa de Jornet 38
06800 mérida (spain)
jlguisado@unex.es

Daniel Lombraña González
Centro Universitario de Mérida
C/ Sta Teresa de Jornet 38
06800 mérida (spain)
daniellg@unex.es

Miguel Cárdenas Montes
CETA-CIEMAT
Paseo Ruiz de Mendoza 8
10200 Trujillo (Spain)
miguel.cardenas@ciemat.es

F. Fernández de Vega
Centro Universitario de Mérida
C/ Sta Teresa de Jornet 38
06800 Mérida (Spain)
fcofdez@unex.es

Abstract

In this article an experience of the utilization of PRC (Public Resource Computation) in research projects that needs large quantities of CPU time is presented. We have developed a distributed architecture based on middleware BOINC and LilGP Genetic Programming tool. In order to run LilGP applications under BOINC platforms, some core LilGP functions has been adapted to BOINC requirements. We have used a classic GP problem known as the artificial ANT in Santa Fe Trail. Some computers from a classroom were used acting as clients, proving that they can be used for scientific computation in conjunction with their primary uses.

1: Introduction

The success of the SETI@Home project [10] has obtained the scientific recognition to bloom multiple projects of Voluntary Computation. These projects possess a wide community of voluntary users that give part of their resource computers. In the area of Distributed Computation we have a wide variety of strategies for building these environments. From stable infrastructures with a high complexity and strong requirements derived from the nature of the problems that they try to study, to lighter applications. A typical example of these first infrastructures is the World LHC Computing Grid (WLCG) [1] destined to

analyze the data provided by the experiments installed at CERN [2] (European Organization for Nuclear Research). Another example is the infrastructure that have born from the EGEE project [3] (Enabling Grids for E-sciencE) and the related sub-projects like EELA [4] (E-infrastructure shared between Europe and Latin America). In the last both examples, the middleware that is being used is gLite [5], which is based on the Globus Toolkit [6].

The Globus Toolkit is an Open Source software used for building Grid infrastructures. The Globus Alliance and many other collaborators in the world are the developers (see [7]).

In all previous cases, the ultimate aim and challenge is to construct a Grid. Though Grid's concept stays out of the scope of this article, a brief definition is provided: A Grid is an infrastructure of computation that re-joins heterogeneous resources distributed geographically through some middleware that allows this mutual interaction. For the construction of Grids several approximations are possible [12]. The infrastructures like WLCG, EGEE or EELA could be interpreted as cluster of clusters, where geographically disperse clusters are linked via advanced communication networks and software infrastructures, thus permitting interoperability between them. On the opposite site, in projects as SETI@Home the construction is done in a completely different approach.

The Grids constructed for the Voluntary Computation (VC) use BOINC [8, 11] (Berkeley Open Infrastructure for Network Computing) as the middleware, being their levels of commitments much slighter. This feature added to the simplicity of operation and the aptitude to coexist in the same computational resources with other uses, have brought a tremendous popularity to this technology. The projects of Voluntary Computation receive diverse denominations in the scientific literature shuch as P2P Computing, Desktop Computing or PRC.

The University of Extremadura and the CETA-CIEMAT proposed to use this Grid platform for providing computational resources to the regional researchers community. In order to realize a proof of concept, we have used a test application consisting of a Genectic Problem developed under LilGP [9] tool.

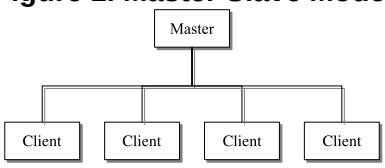
This paper is structured as follows: Section 2 presents Parallel GP. Section 3 describes the adaptation of the LilGP tool to enable it for use under BOINC architecture. Section 4 deals about some planned future works. Finally in section 5 results and conclusions are presented.

2: Parallel Genetic Programming

Genetic Programming (GP) [15] is a metaheuristic inspired by natural evolution of species. This technique uses programs –chromosomes– that will evolve through the time –generations– trying to solve a given problem. The chromosomes are coded as trees and each of them represent a specific program offering a different way of solving the problem. Evolving and evaluating those trees is a heavy task that usually requires huge computational resources. Moreover, chromosomes tend to increase their sizes along the evaluations, so more memory and computational resources will be needed.

An effective way of lightening this load is by using any kind of parallelism inside the algorithm. There are different methods described in literature (see [13]) to parallelize the

Figure 1. Master Slave Model



algorithm. Literature describes two main approaches:

- Individual Level. In this case the individuals will be evaluated in different computers/nodes. This model uses an architecture of Master/Slave (see Fig. 1). The Master node will generate the random initial population and after that it will send the new offspring that have not been evaluated to the Slave nodes. The Slave nodes will evaluate the individual and send back the fitness values to the Master node.
- Population Level. In this case we will have several semi-isolated populations – islands– distributed over different computers/nodes. This technique enables that each island evolves in a totally different way because we will have different copies of the main algorithm in each computer or node. Additionally this technique specifies a migration rate between islands (also known as "demes"). Every population will select a number n of the best individuals and will send them to nearby populations. There are different methods of setting up how and how many individuals will be interchanged between islands.

In this research we have used the Population Level, or the Island Model approach. The first approach will be the use of one island without migration. This is equivalent to execute the same experiment n times in parallel and running the n-times experiments in the same machine.

As said before, LilGP [12] tool is used. This tools addresses the creation of GP problems by using some skeleton files and a set of given functions to build the desired terminal, nodes and fitness functions. The programs generated are ready to accept genetic parameters such as generations, chromosome number, mutation rate, etc. As we have said before we have used a well known framework for Genetic Programming problems, LilGP [12].

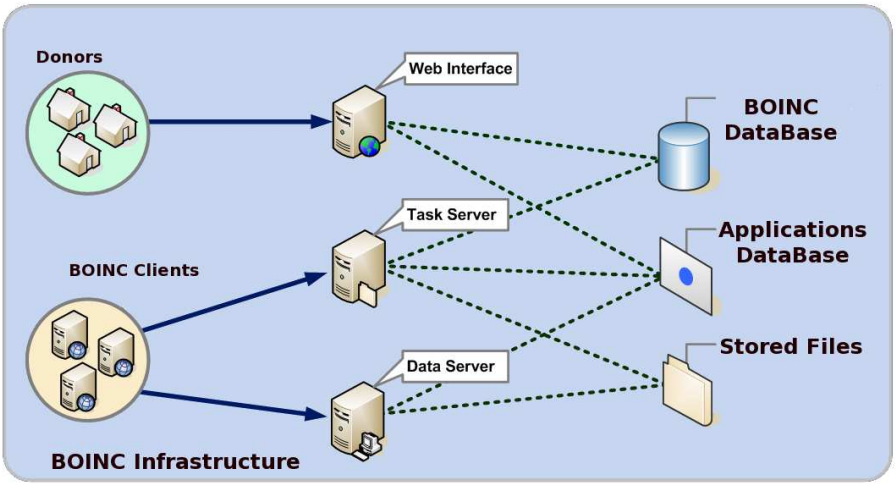
Some adjustments has been made to LilGP core to run under BOINC architecture and in this article that modifications are presented. So an effective integration of LilGP in the BOINC architecture is showed.

3: BOINC Infrastructure

The paradigm of PRC is based on using a heterogeneous set of computers interconnected across Internet: independent computers or belonging to a certain organization.

The computing unit under BOINC is called a project. Every BOINC project is identified by an unique URL, which represent the user start point of the application. The volunteer uses register to projects using that main URL, and get ready to contribute with it. An important challenge for the researcher is to make the project interesting to get as many volunteer users as possible.

Figure 2. BOINC Infrastructure



Other paradigm of use, such as the use of classrooms or administrative PCs doesn't need attract the volunteers because the exploitation of resources is designated by an official decision.

The BOINC infrastructure is composed both on a server and a client side. The server consists of several daemons: a work manager, a scheduler, a feeder, etc. That daemons can be installed on a single server configuration or on a multiple server configuration. On the other hand a set of clients are connected to the server.

The different components of the BOINC infrastructure and the interrelationship between them appear in the Fig. 2. The components in dark tone are provided as part of the system BOINC, whereas the rest must be developed for the managers of the application.

In the server side, BOINC database stores information about the different project components:, which are used in the project: applications, platforms, versions, workunits, results, users, teams, etc. The server generates the work units and sends all of them to the clients. The task server handles the sending of the applications and input files to the clients, and the reception of the results. The Web server provides the interface to the volunteer people. It includes a standard set of pages plus the pages created by the managers. All components in the server are controlled by BOINC software engine.

The client receives the works and input files from the server, executes the different works and sends to the server the results. The BOINC client consists on a little framework that realises the mentioned tasks and an executable to perform the works. That executable must be developed for the managers to realize the. The executable of the application communicates with the framework BOINC using the BOINC API.

As it has been mentioned, all services can be placed physically in the same computer, or can be dispersed in several computers. This fact is strongly correlated with the size and maturity of projects. Usually the evolution of project forces them to store and to handle higher data volumes, to send a bigger number of tasks to clients, and so on. In these cases, the easier way to reach higher commitments is to split the functionalities of server between several computers.

4: LilGP and BOINC

LilGP it's a classic tool used for developing Genetic Programming applications. With the appropriate changes, it's possible to adapt it to the BOINC premises, being capable of generating GP applications that could be executed in BOINC project.

LilGP consists of a core that combines the generation of initial populations for a given problem, together with the operations required for crossing over chromosomes, mutating them, etc.; all of them proper of the Evolutionary Algorithms. This tool is written in C and was created originally to be compiled with gcc, nevertheless, the tools which will serve the project BOINC must be written in C++ and compiled with g++. Due to the difference of versions of the compiler used by LilGP and the tools which are compiled by libraries BOINC, it is necessary to adapt the first one in order that it could be compiled with g++, using in turn BOINC's specific libraries. Once adapted LilGP's code under BOINC's premises, the new tool is ready to work employing BOINC technology for solving GP problems.

Other of the modifications over the LilGP core is related to the I/O operations required for generating the LilGP result files. Due to BOINC's philosophy, several executions of the same program take place on the same client; this may cause unwanted over-writings along the different executions of the same program. BOINC approaches this problem by generating unique random names for each file of each execution, so accidental writings are avoided. The same method is applied for storing the results originated by different clients. The execution of the GP application in a BOINC client generates the same result files as an execution on a local machine. Our I/O treatment applied to LilGP core deals with this issue, using BOINC API to make correspondence between LilGP nature file names and unique random BOINC filenames.

In [14] an adjustment of GP's classic problem is presented. The problem adapted to BOINC was the Artificial Ant on Santa Fe Trail. The source code of the previously mentioned problem was modified attending to the premises for applications which will be executed in BOINC clients. The idea was also to measure the performance offered by the new BOINC based tool when compared to traditional sequential executions in a single computer.

5: Experiments and results

The Ant Problem has been used with the experiments detailed below. The experiments have been made with a Pentium IV PC, 2.8 GHz and 512 MB acting as BOINC server. The hardware specifications for the clients were the same. 25 executions of the experiment have been performed. That number assures certain quality of the results because Genetic Algorithms are a stochastic process, which needs a big number of executions in order to offer statistic consistency for the values. Firstly, a unique experiment with 5 executions has been made in order to study the evolution when sending one task per client. Then, the same experiment with 10 executions was made.

	Tasks (n)	Seq. time (s)	BOINC time	BOINC acc
50 G, 2000 C	5	130s	307s	0.0977
1000 G, 2000 C	25	650s	395s	1.6456
1000 G, 1000 C	25	4250s	1548s	2.7455
2000 G, 1000 C	25	9200s	2356s	3.9049

Table 1. Results using 5 machine clients.

	Tasks (n)	Seq. time (s)	BOINC time	BOINC acc
1000 G, 1000 C	25	4250s	1033s	4,1142
2000 G, 1000 C	25	9200s	1623s	5,6685

Table 2. Results using 10 machine clients.

The Table 1 shows the generation number (G) per task and chromosome number (C) per generation. The obtained results show the execution time in sequential mode (t-sec) against the execution time in BOINC architecture (t-BOINC). The obtained acceleration is calculated by the formula

A = (t-sec)/(t-BOINC)

When only few generations is used, regardless of chromosome number, the obtained results are worse when using BOINC computating face to sequential mode. When the generation number grows, maintaining the chromosome number constant, the acceleration number begins to arise. Table 2 shows an important growth when more clients are added.

Launching the Ant Problem under BOINC architecture better results are obtained than using a sequential platform. In an experiment with 50 generations, 2000 chromosomes and 5 executions the acceleration is 0.0977. With 25 executions this acceleration is 1.6456. With 1000 chromosomes, 25 executions and 1000 generations in first case, 2000 generations in second case, the acceleration grows significantly, obtaining values of 2.7455 and 3.9049 respectively. As was expected, light task with high communication costs makes the sequential model more appropriated than the BOINC one. Nevertheless, in big tasks with a client number growing up, an important acceleration growing can be noticed: the last experiment in table 2 shows an acceleration of 5.6.

6: Future Works

A new version of our LilGP-BOINC is being implemented by the authors. The main modification consists of implementing the island model with migration of individuals. To make possible this modification we need a common point processable both by the BOINC server and the LilGP core. The LilGP tool with the ant problem was adapted again for emplying checkpoint files. These files allow to store the intermediate state of the popu-

lation after computing a given generation, or the state of the population at the end of the execution instead.

As soon as the executions in the client finish, the information is returned to the server, where a process made in conjunction by the validator and the assimilator will perform the migration of the individuals between the different populations sent to every client. The next unit of work generated by the server will send to every client a new population, composed by individuals of different populations, previously combined by the validator and the assimilator.

The challenge of implementing migration between islands is complex because of the nature of the BOINC architecture. That architecture, based on a server and a number of clients communicating with it, doesn't allow communication between clients so we can not transfer data from any given client to another one. Instead, we have to simulate that communication by processing the results in the server.

That processing consist of receiving the population files (or checkpoint files) explained in previous sections from the clients. Then, an interchange of the lines of these files is made to perform migration. At the end of the interchange process we will have the resulting population files for the next generation with their best individuals migrated. The next step is to order the new files by fitness. Finally it's necessary to create new BOINC workunits using the new files as the input. The BOINC engine will send that files to the clients in order to the genetic process can continue.

To perform this process authomatically we will use the BOINC validator. The validator is a BOINC component consisting on a daemon which acts when a result from a client is received and takes the action given by the programmer to check if the incoming results are valid ones. The validator is proggramed in C code, so in addition to this checking we will use it to launch the interchange process. Once interchanged, the new population files are the starting point for the next generations. In the final generation, where no migration is needed, the validator takes no action and the final results are stored by the assimilator and becomes available for the scientist.

7: Conclusions

In this paper, an experience in using desktop grid computing for research projects with large runtime requirements has been presented. PRC has been used for this approach. The BOINC software platform for distributed computing has been used. The LilGP Genetic Programming tool has been modified to use the BOINC platform. By using this modified version of LilGP, the developing of GP applications distributed on BOINC architecture is quick and easier.

As a proof of concept, a typical Genetic Programming problem (the Artificial Ant on Santa Fe Trail) has been run on classroom computers using the LilGP-BOINC framework. That framework will be expanded to enable island model and migration. The experiments made shows that applying the distributed benefits of the BOINC platform improves the quality of the results The possibility of using existent computing resources from an organization for scientific computing with this kind of framework, without perturbing the main usage of the computers, has been demonstrated.

References

[1] WLCG, (World LHC Computing Grid), http://lcg.web.cern.ch/LCG/.

[2] European Organization for Nuclear Research, http://www.cern.ch/.

[3] Enabling Grids for E-Science, http://www.eu-egee.org/.

[4] E-infrastructure shared between Europe and Latin America, http://www.eu-eela.org/.

[5] gLite, http://glite.web.cern.ch/glite/.

[6] Globus Toolkit, http://www.globus.org/toolkit/.

[7] Globus Alliance, http://www.globus.org/.

[8] BOINC, (Berkeley Open Infrastructure for Network Computing), http://boinc.berkeley.edu/.

[9] LilGP, http://garage.cse.msu.edu/software/lil-gp/.

[10] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, 2002.

[11] D.P. Anderson. Boinc: a system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10, 2004.

[12] Miron Livny Douglas Thain. *The Grid 2*, chapter 19, pages 285–318. Morgan Kaufmann, 2004.

[13] M. Tomassini. *Spatially Structured Evolutionary Algorithms*. Springer, 2005.

[14] F.Chávez, J.L. Guisado, D. Lombraña González, F. Fernández. Una Herramienta de Programación Genética Paralela que Aprovecha Recursos Públicos de Computación. In *V Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados*, 2007.

[15] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.