A Hybrid ECJ+BOINC Tool for Distributed Evolutionary Algorithms

Francisco Fernández de Vega[1], Leonardo Trujillo[2], Francisco Chávez[1], Enrique Mediero[1], and Luis Muñoz[2]

[1] Universidad de Extremadura, C/. Santa Teresa de Jornet, 38, 06800, Mérida, España
fcofdez@unex.es, fcchavez@unex.es, enmediero@gmail.com
[2] TREE-LAB, Posgrado en Ciencias de la Ingeniería, Instituto Tecnológico de Tijuana, Av. Tecnolígico S/N, Fracc. Tomós Aquino, Tijuana, B.C., México
www.tree-lab.org
leonardo.trujillo@tectijuana.edu.mx, lmunoz@tectijuana.edu.mx

Paper received on 11/21/13, Accepted on 01/19/14.

Abstract. This paper presents an improvement to ECJ (Evolutionary Computation in Java), the popular evolutionary computation tool, which allows users to exploit distributed computational resources through the use of volunteer computing. In particular, the BOINC (Berkeley Open Infrastructure for Network Computing) middleware is used to distribute ECJ client software on top of a virtualization layer, this allows researchers to parallelize their experiments without having to port their software to make it compatible with BOINC. In this way, an interested researcher can use the ECJ+BOINC software in the same way that they use the standard ECJ tool. Moreover, the system allows the user to choose between different distribution models based on their preferences. Finally, the ECJ+BOINC system is developed in a modular manner, allowing for easy updates and modifications based on the possible requirements of future ECJ versions.

Keywords: Evolutionary Computation, Distributed Computation, ECJ, BOINC

1 Introduction

Every year, the importance of distributed and parallel computing models becomes more apparent, with potential impacts in all of computer science. Recently, the president of the IEEE Computational Intelligence Society, in the May 2013 issue of the Computational Intelligence Magazine, listed a series of challenges that must be addressed by researchers in the field of Computational Intelligence. In particular, two problems stand out, commonly referred to as *Big Data* and *Real Time* [1]. For most researchers, these terms suggest the need to process large amounts of data on the one hand, and the need for fast and efficient processing on the other. It is also relevant to mention that from the perspective of Evolutionary Computation (EC), a sub-field of Computational Intelligence, both concepts are not part of the canonical formulation of evolutionary algorithms (EA) for search and optimization [2].

Over recent years many proposals have attempted to integrate notions of parallel and distributed computation within traditional EA, such as parallel EA models or traditional algorithms that are simply parallelized at the implementation level. However, despite such work, there is still a tendency for researchers to use sequential models and implementations, what is commonly referred to as immobilization within the software industry. In general, researchers prefer to use well-known computational tools before investing time and resources in modifying existing tools or developing new ones.

Therefore, some works have focused on developing computational tools that facilitate the use of distributed resources while keeping the amount of time and effort at a minimum. For instance, in [8] and [6] the authors proposed a model that emphasizes the cloud computing model and allowed users to deploy virtual machines that internally executed an evolutionary algorithm within a volunteer computing setting. While the proposal met the stated goals, it suffered from two possible shortcomings: (a) the size of the distributed virtual machines was quite large (several gigabytes); and (b) researchers needed to prepare and deploy the necessary infrastructure to run the Berkeley Open Infrastructure for Network Computing (BOINC) middleware [4].

The current work goes one step further, by considering one of the most well-known and widely used EC tools called Evolutionary Computation in Java (ECJ) [5], and integrating within it the BOINC technology. The goal is to allow any researcher that is familiar with ECJ to launch experimental distributed experiments based on BOINC. Moreover, another goal is to make the required learning curve of the new technology less severe; this is accomplished by fully integrating a BOINC server within ECJ, and the necessary virtual machine within the BOINC client. In this way, an ECJ user only has to perform two simple tasks: (1) run the virtual machine that acts as server, by compiling the problem specific EA within this server just as it is normally done in ECJ; and (2) run the client virtual machines. When the EA is executed, the ECJ+BOINC system automatically distributes its execution across all available distributed resources or nodes.

This paper presents a proof-of-concept implementation of the ECJ+BOINC system, and also outlines specific areas for future work and improvements. The remainder of this paper is organized as follows: Section 2 reviews related work in volunteer and distributed computing models for EAs. Then, Section 3 describes the proposed methodology. Afterwards, experiments and results are summarized in Section 4. Finally, concluding comments and future work are discussed in Section 5.

2 Volunteer Computing and Distributed EAs

Volunteer computing, a specific distributed computing model, has gained large acceptance over recent years. It is based on users voluntarily cooperating with research projects, by offering their computing resources (personal computers or other devices) that are connected to the Internet to act as data processing units. The model uses specialized software that exploits computing and storage resources that normally would be wasted, and assigns these resources to collaborative research projects [7]. The most widely used and successful middleware for this task is BOINC [4]. Currently there are more than one million volunteers collaborating worldwide on BOINC projects, donating their resources and CPU cycles[3].

In particular, the BOINC model is now also used with some EAs, achieving interesting results when evaluated based on system robustness and fault tolerance [6]. Therefore, there is a current interest in further exploiting such models in EC projects and applications, that normally incur high computational costs and need to process and store large amounts of data.

BOINC was developed two decades ago with the *seti@home* project, providing all of the basic software tools required to deploy a volunteer computing system or project. It includes a BOINC server, that distributes the computational load among a set of volunteer client machines that are connected to a specific project. Moreover, it also includes a BOINC client, that is in charge of communicating with the server and executing the computational tasks assigned to the client machine by the server. While the general model is simple and straightforward, particularly compared to other GRID-based models, to use the basic BOINC tools on a specific project, the source code must first be ported to operate within the BOINC environment through the provided API. This can be a big limitation in some instances, since not all programming languages are supported by the BOINC API, sometimes requiring a large scale re-coding project that will require some IT management experience in distributed computing environments. Moreover, in some cases it simply is not possible to port a particular system that has strong dependencies with specialized software or simulators; for instance, one can imagine such a case for an application that requires Matlab or Mathematica.

To overcome these and other possible problems, some researchers have developed partial solutions based on virtualization, in a sense encapsulating projects before distributing them over a BOINC network of clients [6]. However, this model imposes limits over the distributed computing paradigm: it is only possible to execute a single application multiple times. For instance, it is not possible within a traditional EA, to distribute individual candidate solutions for fitness evaluation over BOINC or perform load balancing, when fitness evaluations are not homogeneous, a common problem in genetic programming (GP) systems for example.

[3] http://boinc.berkeley.edu/

In summary, while the technology is currently widely available, and some partial simplified solutions have also been developed, there is still a lack of a complete system capable of integrating the BOINC middleware within an evolutionary search. This paper presents such an alternative, the first attempt at integrating the BOINC advantages with an EC toolkit.

2.1 ECJ & BOINC

ECJ is a very popular and well-known tool used by researchers within the Computational Intelligence community, with many works publishing results that either use ECJ or that are compared with the tools provide by ECJ. However, there is no off-the-shelf way of distributing an ECJ application through BOINC.

While ECJ does offer some parallel computing models to exploit several processors or cores, it is not possible to launch an ECJ algorithm within a distributed BOINC environment. This work focuses on one of the simplest possible models, where individuals from the evolving population are distributed by BOINC to be evaluated by a set of connected clients. To achieve this, ECJ must be modified accordingly, which is the contribution presented in this work.

Another goal is to develop software tools that simplify the use of such a distributed model for an EA. Therefore, a BOINC server is integrated within ECJ, configured to distribute work units composed of sub-sets of individuals from an EA developed with ECJ, to clients that are configured and connected with the server. Moreover, to further simplify the process for the user, all of this was encapsulated within dedicated virtual machines, allowing researchers to simply download the virtual machine server and run it on the server machine. Then, the user must define its fitness function and other problem-specific details of their ECJ project, and include these functions in the sever before it is compiled, like any other ECJ applications. When the application is finally executed, the work units and all of the BOINC middleware are configured and launched automatically. At the other end, client machines must download the BOINC clients, and connect to the BOINC server to participate with a project. The client-side virtual machine is configured to automatically connect with the server-side virtual machine, without requiring any additional administrative or configuration tasks by the user. In what follows, the proposed system is described in detail, referred to as the ECJ+BOINC system.

3 Proposal

As stated before, the goal of this work is to develop the necessary software tools that allows a researcher to exploit the BOINC paradigm of distributed volunteer computing with EAs written with the ECJ toolkit. In what follows, the underlying technical implementation of the proposed system is described, focusing on each of the technologies that were employed.

3.1 Virtualization

To start, lets consider the basic requirements to be able to incorporate BOINC within ECJ. As stated before, there are two main pieces of software that need to be included: (a) a BOINC server that distributes the work units; and (b) the BOINC clients that perform the requested computations. Both software modules, the server and clients, are distributed over a virtualization layer, in this case VirtualBox [4]. This virtualization layer can be used by clients running on different operating systems, it is simple to setup and install.

The virtual machines are configured with Linux, along with all of the necessary software tools to administrate the server and client tasks independently. Moreover, these virtual machines also have all the required software to run ECJ applications, both on the server and the client.

3.2 ECJ

ECJ is probably the most popular EC tool, developed by Sean Luke et al. ECJ is written in Java and includes a large set of features and state-of-the-art methods [5]. It was designed to be flexible and highly configurable, with almost all of its parameters determined at run time from a specified set of configuration text files.

The structure of ECJ includes a large set of specialized classes, among them the most relevant to the current work is the *Evolve* class. Among others, the *Evolve* class includes the *main()* method, which is executed to start all evolutionary algorithms implemented in ECJ. Another noteworthy class is *Evaluator*, to which the task of evaluating an evolving population is assigned, of particular importance for a system that distributes population evaluation through BOINC. As stated above, the client machines will evaluate subsets of the evolving population, hence each client needs to employ an *Evaluator* object, that returns the necessary output to the server, to guide the search process being executed by an *Evolve* object on the server.

3.3 BOINC

BOINC is written and developed in C/C++, thus it most easily integrates with systems written in the same programming language. Since not all projects that are deployed with BOINC are written in C/C++, it is necessary to port the application or employ specialized *Wrapper* objects. Through a wrapper object or function, it is possible to run non C/C++ code within a BOINC client. On the other hand, the BOINC architecture is based on the client-server model that allows for multiple *work units* to be deployed simultaneously by the server, which are then processed on the BOINC clients, while the server waits for the results returned from each client. The server can validate the results returned by the clients, and determines if the results are accepted or rejected, in which case the corresponding work unit can be sent to another client.

[4] https://www.virtualbox.org/

3.4 ECJ+BOINC

To integrate BOINC with ECJ, it is necessary to modify the normal execution sequence of a BOINC project, since part of of the ECJ application is kept on the server side, and only population evaluation is distributed over BOINC. Normally, in a BOINC project the server only generates and distributes work units, it then waits idle while the clients perform all of the data processing associated with a project. On the other hand, in this work the server is in charge of executing an EA, and only during fitness evaluation the EA is paused on the server side. At this moment, the server generates a series of work units that are sent to the BOINC clients, each work units contains a subset of individuals from the population that must be evaluated on the client side. Once the clients compute the corresponding fitness values, these are returned to the BOINC server, which then continues with the normal EA execution.

This program flow also requires new code for the ECJ classes used during evolution. First, a new class has been derived from the ECJ *SimpleEvaluator.java* class, by adding a Shell Script that generates a single work unit, intended for a single BOINC client; this is the simplest possible case. The server then waits idle, until the results are uploaded onto the *upload* directory of the corresponding project.

Once program execution is halted on the server, the current state of the project run is saved using the *checkpoint.java* class. This class is used to be able to send the current state of the run to the client, along with the population that will be evaluated. The client evaluates the population, and updates the state of the current run, which is returned to the server so the search can continue. Figure 1 presents the basic program flow of the ECJ+BOINC system.

Based on this simple single client version, the next step is to extended the ECJ+BOINC system to run with several clients, each assigned the responsibility of evaluating a subset of individuals from the population.

Similarly to the simpler case, once ECJ execution is paused on the server, the requested number of work units are generated for each client; note that not all clients will necessarily evaluate the same number of individuals. Once each client terminates its evaluation process and the results are returned to the server, the partial results of each client must be integrated into a single state file that updates the state of the EA running on the server. This requires the following modifications to the ECJ library:

- Class *EvaluatePopulation*: before the population is evaluated, the state of the EA is saved.
- Class *evalPopChunk*: calls a process that creates the requested work units and distributes them to the clients.

Once the clients have returned their results normal ECJ execution continues on the server, as depicted graphically in Figure 2, where ECJ+BOINC is running with x number of clients.

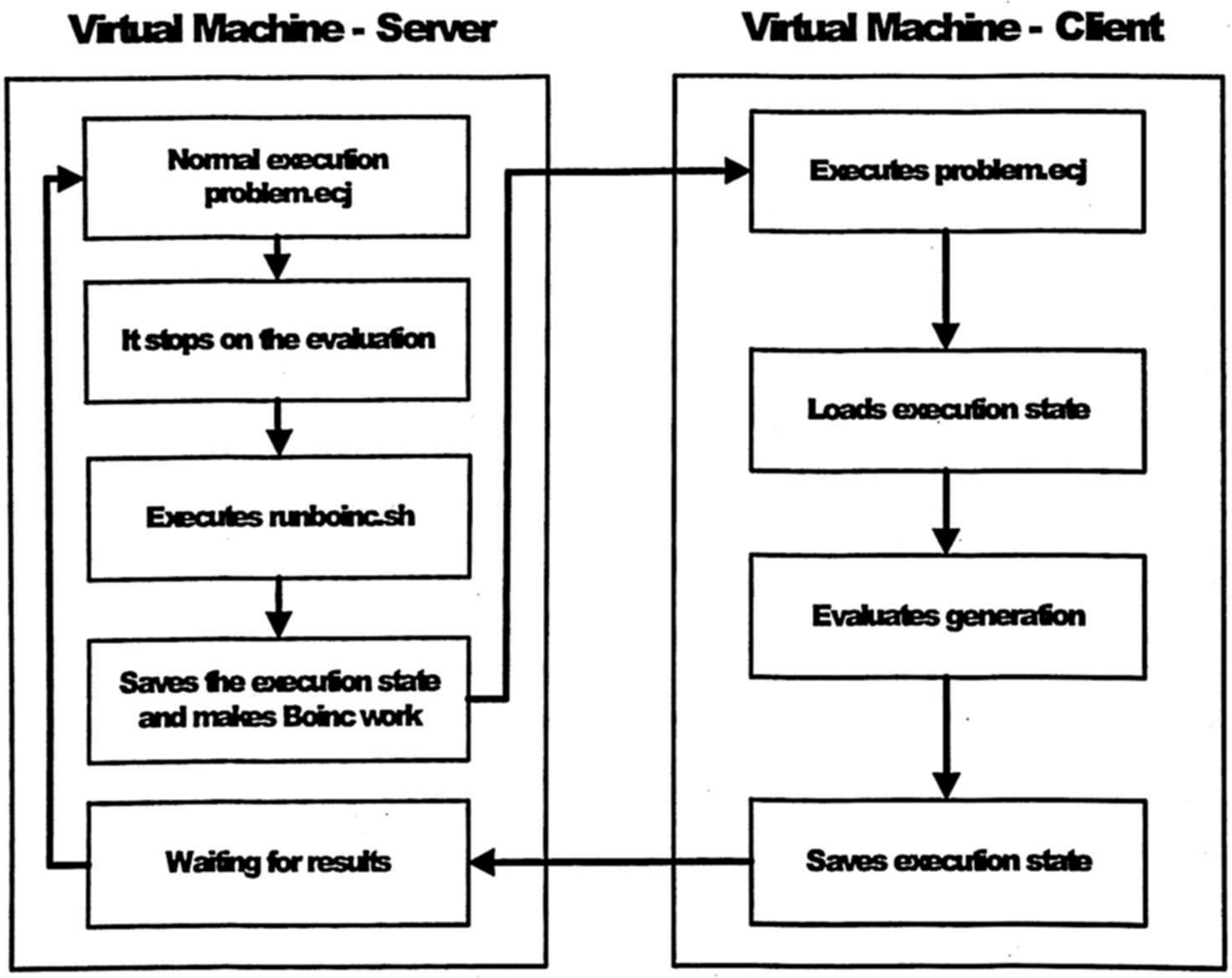


Fig. 1. Program flow of an ECJ+BOINC system running with a single client.

4 Experiments and Results

To test the proposed ECJ+BOINC system, this work uses a typical GP benchmark problem with a high computational cost, the Even Parity problem [3]. This problem is widely used for benchmarking purposes, however instead of using the common 5-bit problem, the number of bits was set to 30, increasing the difficulty of the problem and forcing the system to incur higher computational costs during fitness evaluation. In this scenario, a normal ECJ GP run required approximately 20 minutes to evaluate the fitness of 10 individuals on a standard PC.

Besides the small number of individuals used during the run, 10, and the relatively small number of generations set to 25, all other parameters were set at the default values provided by ECJ. Indeed, the goal of the tests are not to evaluate if the GP algorithm provided by ECJ can solve this problem, this is irrelevant to the tests, the goal is to illustrate the performance gains provided by BOINC-based distribution of fitness evaluations. To test ECJ+BOINC, five clients were connected to the server machine, on which the virtual machine client and BOINC client were installed. During fitness evaluation, a single individual

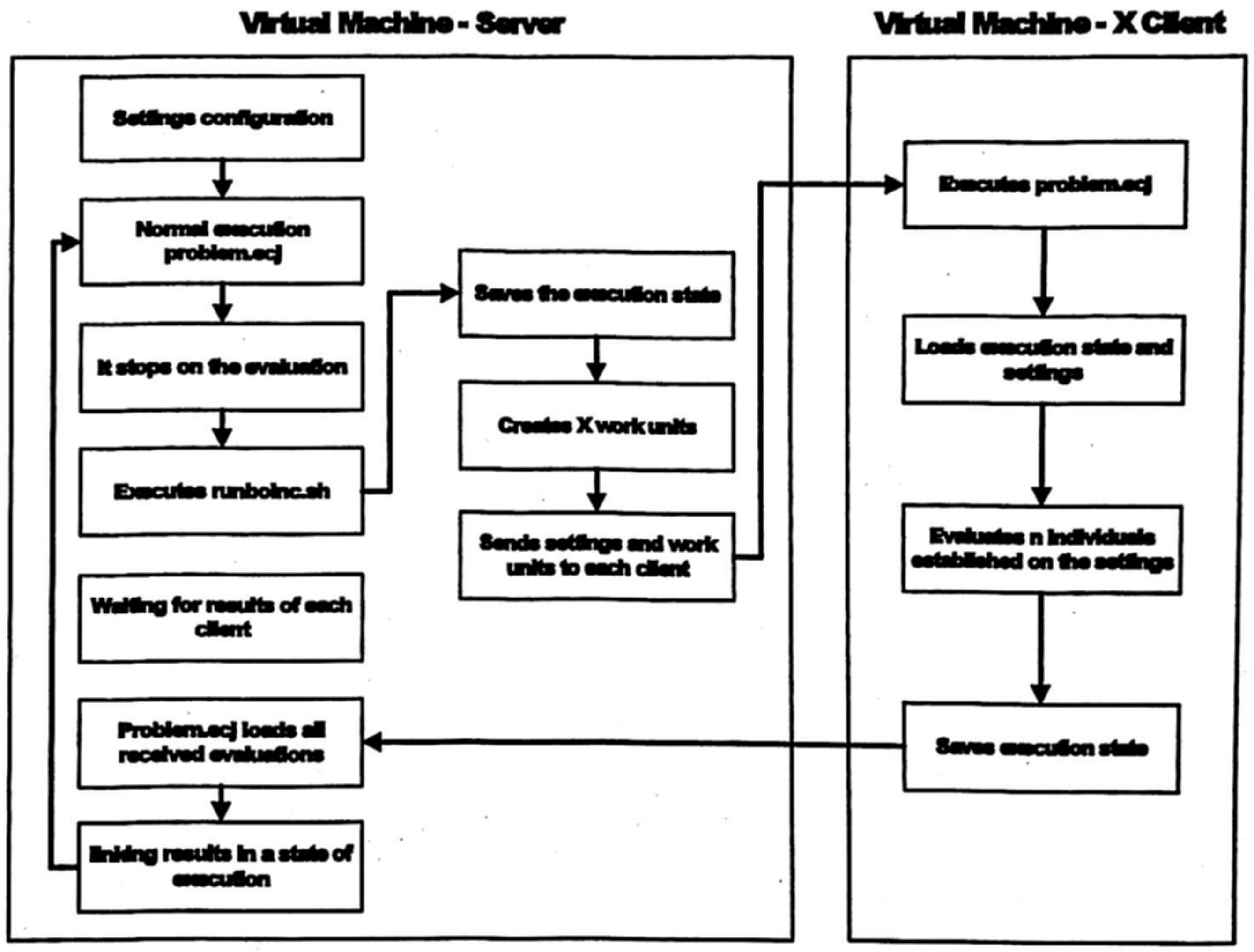
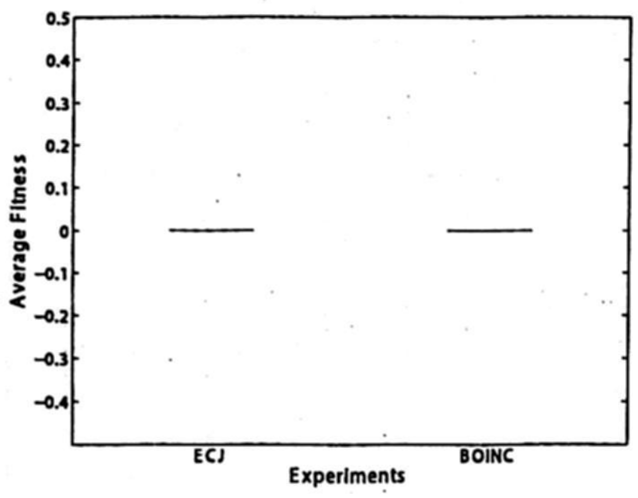

Virtual Machine - Server
Virtual Machine - X Client
Settings configuration
Normal execution problem.ecj
It stops on the evaluation
Executes runboinc.sh
Waiting for results of each client
Problem.ecj loads all received evaluations
linking results in a state of execution
Saves the execution state
Creates X work units
Sends settings and work units to each client
Executes problem.ecj
Loads execution state and settings
Evaluates n individuals established on the settings
Saves execution state

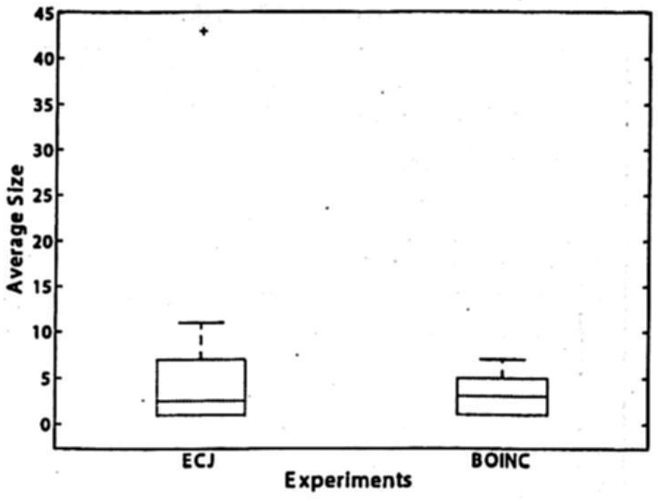
Fig. 2. Program flow of an ECJ+BOINC project running with x number of clients.

is distributed to each client at each generation. Ten runs were executed, using standard ECJ and the ECJ+BOINC system.

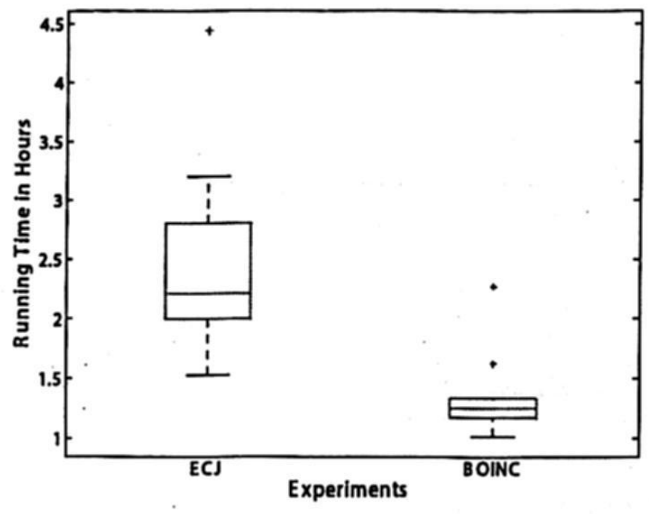
Figure 3 summarizes the results of these experiments in boxplots comparing the ECJ system with the ECJ+BOINC configuration. First, Figure 3a shows that there is no difference between both systems based on the quality of the results found. This is desired, the underlying BOINC distributed model is not designed to modify or alter the search dynamics, its sole purpose is to reduce computation time by distributing costly fitness evaluations. Similarly, the fact that ECJ+BOINC does not effect search dynamics is confirmed by Figure 3b, that shows the average program size evolved in each GP run. Again, ECJ and ECJ+BOINC basically produce the same results, which means that the BOINC-related modifications did not inadvertently modify the search process. Finally, Figure 3c summarizes the results related to the run time of each system. As expected, ECJ+BOINC clearly reduces the computation time required to evolve solutions for the benchmark problem, basically reducing the median run time in half.



(a) Average Fitness



(b) Average Program Size



(c) Run Time

Fig. 3. Summary of the experimental comparison between ECJ and ECJ+BOINC.

5 Summary, Conclusions and Future Work

Overall, distributed and parallel computing has a large impact every research area where large amounts of data need to be quickly processed. One way to obtain the computational power and storage resources required to solve such problems, is to exploit the volunteer computing model popularized by the BOINC middleware in many real-world projects. In general, however, BOINC has still not been fully integrated within EC systems, that almost always need to improve efficiency and reduce run times.

This work represents the first attempt to integrate BOINC into a popular and widely used EC tool, the well-known ECJ library. In this work, BOINC is used to distribute the population over a set of connected clients, basically parallelizing fitness evaluation during an ECJ run, normally the most severe bottleneck in an EA. Moreover, this was done in such a way so as to make the entire process transparent for an ECJ user, who has to do little additional configurations or setup compared to a basic ECJ run. This is achieved by deploying the BOINC clients and server within specialized virtual machines, something that is automatically accomplished by the ECJ+BOINC system.

Results show that the proposed solution integrates nicely with ECJ, without affecting the search process in any way, only producing a substantial improvement in run time on a benchmark problem for GP. While these initial results are encouraging, future work is still requiered. For instance, future research should focus on employing the ECJ+BOINC system on a real-world problem, where fitness evaluation is more costly, for instance with evolutionary robotics, where fitness depends on a costly simulation process [9]. Moreover, in such a case the use of a virtual machine will be even more beneficial, easily deploying a large amount of fitness-specific software libraries over a large set of distributed clients. Finally, it will be of interest to explore how a distributed system, such the one presented in this paper, can be used to enhance the search dynamics of an EA [10, 11]. These algorithms could produce interesting epiphenomenons, such as reducing bloat in a GP-based search [12].

6 Acknowledgments

Funding for this work provided by Ministerio de Ciencia e Innovación, through the ANYSELF project (TIN2011-28627-C04), Universidad de Extremadura and Gobierno de Extremadura Consejería de Economía Comercio e Innovación and FEDER project No. GRU10029. Additionally support provided by CONACYT (Mexico) Basic Science Research Project No. 178323, DGEST (Mexico) Research Projects No.5149.13-P and TIJ-ING-2012-110, and IRSES project ACoBSEC from the European Comission. Fifth author supported by CONACYT scholarship No. 302526.

References

1. Polycarpus, M.: Computational Intelligence in the Undergraduate Curriculum. Computational Intelligence Magazine, 8:2 3 (2013).
2. De Jong, K.: Evolutionary Computation: A Unified Approach. The MIT Press. (2001)
3. Koza, J.R.: Genetic Programming. MIT Press. 1992.
4. Anderson, David P.: BOINC: A System for Public-Resource Computing and Storage, Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing. GRID '04 4–10 (2004).
5. David R. White: Software review: the ECJ toolkit, Genetic Programming and Evolvable Machines 13:1 65–67 (2012)
6. F. Fernández de Vega, G. Olague, L. Trujillo, D. Lombraía: Customizable execution environments for evolutionary computation using BOINC + virtualization, Natural Computing 12:2 163-177 (2013)
7. David P. Anderson: Volunteer computing: the ultimate cloud, ACM Crossroads 16:3 7–10 (2010).
8. Chávez, Francisco and Guisado, Jose Luís and Lombrana, D and Fernández, Francisco: Una herramienta de programación genética paralela que aprovecha recursos públicos de computación, MAEB 167–173 (2007)

9. L. Trujillo, G. Olague, E. Lutton, F. Fernández de Vega, L. Dozal and E. Clemente: Speciation in Behavioral Space for Evolutionary Robotics. Journal of Intelligent & Robotic Systems 64:34, 323–351 (2011)
10. M. García-Valdez, L. Trujillo, F. Fernández de Vega, J.J. Merelo Guervás and G. Olague. EvoSpace: a distributed evolutionary platform based on the tuple space model. In Proceedings of the 16th European conference on Applications of Evolutionary Computation (EvoApplications'13), Anna I. Esparcia-Alcázar (Ed.). Springer-Verlag, Berlin, Heidelberg, 499–508 (2013).
11. M. Garcia-Valdez, J.J. Merelo, L. Trujillo, A. Mancilla and F. Fernandez-de-Vega: Is there a free lunch for cloud-based evolutionary algorithms? IEEE Congress on Evolutionary Computation 2013, Cancun, Mexico, 20 - 23 June, 2013. IEEE Press, 2871–2879 (2013).
12. Robin Harper: Spatial co-evolution: quicker, fitter and less bloated. In Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference (GECCO '12), Terence Soule (Ed.). ACM, New York, NY, USA, 759–766 (2012).