Nat Comput (2013) 12:163–177
DOI 10.1007/s11047-012-9343-8

Customizable execution environments for evolutionary computation using BOINC + virtualization

Francisco Fernández de Vega · Gustavo Olague · Leonardo Trujillo · Daniel Lombraña González

Published online: 28 August 2012

Abstract Evolutionary algorithms (EAs) consume large amounts of computational resources, particularly when they are used to solve real-world problems that require complex fitness evaluations. Beside the lack of resources, scientists face another problem: the absence of the required expertise to adapt applications for parallel and distributed computing models. Moreover, the computing power of PCs is frequently underused at institutions, as desktops are usually devoted to administrative tasks. Therefore, the proposal in this work consists of providing a framework that allows researchers to massively deploy EA experiments by exploiting the computing power of their institutions' PCs by setting up a Desktop Grid System based on the BOINC middleware. This paper presents a new model for running unmodified applications within BOINC with a web-based centralized management system for available resources. Thanks to this proposal, researchers can run scientific applications without modifying the application's source code, and at the same time manage thousands of computers from a single web page. Summarizing, this model allows the creation of on-demand customized execution environments within BOINC that can be used to harness unused computational resources for complex computational experiments, such as EAs. To show the performance of this model, a real-world application of Genetic Programming was used and tested through a centrally-managed desktop grid infrastructure. Results show the feasibility of the approach that has allowed researchers to generate new solutions by means of an easy to use and manage distributed system.

Keywords Boinc · Virtualization · Desktop grid systems · Evolutionary algorithms

F. Fernández de Vega
University of Extremadura, Mérida, Badajoz, Spain
e-mail: fcodez@unex.es

G. Olague
EvoVision Project, Computer Science Department, Centro de Investigación Científica y de Educación Superior de Ensenada, Km. 107 Carretera Tijuana-Ensenada, 22860 Ensenada, BC, México
e-mail: olague@cicese.mx

L. Trujillo (✉)
Doctorado en Ciencias de la Ingeniería, Departamento de Ingeniería Eléctrica y Electrónica, Instituto Tecnológico de Tijuana, Blvd. Industrial y Av. ITR Tijuana S/N, Mesa Otay, C.P. 22500 Tijuana, BC, México
e-mail: leonardo.trujillo.ttl@gmail.com

D. Lombraña González
Citizen Cyberscience Centre, CERN, Geneva, Switzerland
e-mail: teleyinex@gmail.com

1 Introduction

Evolutionary computation (EC) deals with the development of search and optimization algorithms that are based on the core principles of Neo-Darwinian evolutionary theory (De Jong 2001). In computational terms, evolutionary algorithms (EAs) are population based meta-heuristics that apply stochastic search operators and are guided by an objective measure of performance, also known as the fitness function. Evolutionary search relies on a diverse sampling of the solution space, to extract a meaningful description of the underlying fitness landscape to guide the search process towards high fitness peaks.

The convergence of EAs towards optimal solutions can only be guaranteed for a small number of problems (Poli et al. 2007), that is why EAs are mostly used when

approximate solutions are sufficient because globally optimal solutions cannot be derived analytically or found in a reasonable amount of time (Holland 1975). This is particularly the case for problems with complex search spaces and multi-modal fitness landscapes. For instance, such is the case for Genetic Programming (GP), where the goal is to evolve computer programs for specific tasks, what is referred to as automatic program induction (Koza 1992). Therefore, an integral part of EAs is an explorative process that allows the search to locate promising regions within the search space with higher than average fitness values. To do so, EAs normally need to maintain large populations of candidate solutions. This can be computationally expensive, particularly when several fitness cases must be evaluated for each individual and the fitness function takes long. However managing a large population can mostly be solved by appropriate parallelization. In this way, for many real-world problems the biggest computational bottleneck is usually computing fitness. Such is the case when fitness depends on processing large amounts of data (Olague and Trujillo 2011), or when candidate solutions need to interact with complex simulators or real-world environments. In these scenarios, executing a single run can be a slow and time consuming process, a problem that is worsened by the fact that most algorithms must be executed a large number of times, like in the case of using different parameterizations, to reach statistically verifiable conclusions.

To overcome this, many researches have proposed distributed schemes, where the evolutionary process is concurrently executed in multiple nodes (Desell et al. 2010). Other parallel models and technologies can also be applied. For instance, in (Laredo et al. 2011) a fine grained distributed model was proposed, using Peer-to-Peer technology, while (Antonio Nebro et al. 2008; Nouredine Melab et al. 2006) consider the application of grid infrastructures. Other approaches have proposed web-based systems, where fitness evaluation is distributed over multiple clients using a browser-based scheme (Merelo Guervós et al. 2010). However, we are interested here in the coarse-grained approach, where entire populations, or complete experiments, are distributed over a set of processing nodes. Notable works in this line are our previous ones (Lombraña et al. 2007a, b), were we presented a preliminary approach for running unmodified applications within DGC thanks to the employment of virtualization. These proposals form the basis for the research presented in this paper.

2 Problem statement and scope of research

The goal of this work is to both employ available computing resources while also providing EA researchers with a framework that conveniently allows deploying experiments by sharing the computational workload inherent in experimental research, thus reducing time to results. In fact, such is the case for researchers in many fields, not just EC. Currently, scientific applications require an increasingly large number of computing resources, which are usually not available. On the other hand, there is no research institution or company today that can afford the shortage of desktop personal computers for improving the efficiency of their employees when solving everyday duties. Those desktops running commodity operating systems, are regularly used for non-intensive CPU applications like administrative tasks. Despite the advantages of using desktop PCs, when considering efficiency and energy consumption, it is easily noticeable that resources are frequently wasted unnecessarily: those desktops are normally unused for long periods, resulting in a waste of computing power (idle CPU) and storage capacity. Today, this problem is even more relevant due to desktops' impressive hardware capabilities such as multi-core CPUs, high capacity hard disks/memories, and the lack of standard applications that can make appropriate use of all the available resources. The result is that a large percentage of CPU time and energy is wasted.

Some years ago (Anderson et al. 2002) proposed the idea of taking advantage of the unused CPU cycles via the Desktop Grid Computing (DGC) technology. DGC tries to retrieve as much unused CPU power as possible without disturbing the user's desktop PC experience and standard work-flow. Even when the usefulness of this technology has allowed several projects to obtain large computing resources, the number of projects that are being developed under this umbrella is still narrow. One of the main reasons why researchers usually prefer to continue using old-fashioned technologies instead of employing a new one is that it requires changes in the underlying application. Yet, virtualization may be of help.

The virtualization paradigm continues to gain wide acceptance and use (Barham et al. 2003; Sugerman et al. 2001; Nieh and Leonard 2000), showing great potential in a number of problem domains that arise from the heterogeneity of computer platforms: computer consolidation, deployment of applications, security and isolation, to mention a few common examples. For the specific problem that we are addressing, virtualization can be useful to circumvent the common DGC and grid wide variety of platforms: it helps to abstract the different hardware architectures and OSes by providing a common layer for all the platforms.

In this paper we build on the previous work and extend it: the DGC technology is in charge of deploying the virtual machines (VMs) along the desktop clients within a private computer networks (commodity computer laboratories

within a single institution, for instance), while the VMs will run the scientific applications provided by researchers—exactly as the scientist run it in a single computer, without the need of changing the source code. We are thus not proposing a new Parallel model for EC, but transparently allowing to massively run experiments. Additionally, we have created a tool for managing all the DGC resources from a centralized web page, allowing to handle hundreds to thousands of computers from a single access point. Therefore we will obtain the benefits of both worlds: collecting and managing unused computing resources by means of DGC while providing those resources directly to researchers without charging them with any extra fee. To sum up, the contributions of this paper include:

- A new computing model for BOINC systems focused on institutions, and
- an end-user application, called Jarifa, that administers the BOINC resources of an institution.
- A novel approach for creating custom execution environments within a BOINC infrastructure,
- the reduction of coding time when deploying Distributed EAs by allowing to run sequential versions of EAs on Distributed environment without any code re-writing or modification.
- Results on a real-life problems, a computationally costly EA, that confirm the usefulness of the proposal,

We must emphasize that no new parallel model for EA is developed here (nor required). Although running multiple executions of a sequetial EA amounts to a multi-population evolutionary search when populations are isolated, and this has already shown to be useful (Fernández et al. 2003), we don't try here to analyse the model from this point of view. The framework allows to run simultaneously multiple executions of the researcher preferred model. We are thus providing a help to researchers that traditionally employ sequential versions of EA algorithms: they easily can collect available computing resources thanks to DG framework and software tools provided, while also save time when running experiments.

The paper is organized as follows: first, we present the related work, then we describe the framework and experimental methodology employed, and finally we discuss our results and outline the main conclusions derived from the research.

3 Related work

DGC technology relies on a *middleware* to profit from desktop PCs idle cycles. The middleware (Kesselman and Foster 1999) is a software layer that exports the computing and storage resources as an homogeneous computing layer for the programmers. The DGC middleware is built on top of different programming languages and technologies; thus, employing a given DGC implies to accept the benefits and limitations of this technology. For this reason, in order to use a DGC middleware researchers may have to adapt the source code of the application. The adaptation or modification step usually results in a complex task due to the different technologies (different libraries, versions of applications, etc.) that can be employed/required by the scientists to carry out their research. Moreover, researchers' applications and desktop PCs may employ different OSes.

It is obvious that due to the additional modification step the development process will require an extra task and raise the cost for scientists in terms of software maintenance (Pressman 2004). Moreover, sometimes this might not be possible when third-party tools are employed by researchers—source code is not available—or scientists do not have the required expertise for adapting the application.

Summarizing, a researcher that wants to use a DGC middleware may have to carry out some changes in his source code. The main changes may be due to some of the following factors:

- *Programming Language*. The DGC middleware employs a different programming language from the scientific application.
- *OS dependencies*. The scientific application embodies a given OS API.
- *Hardware dependencies*. The scientific application depends on a given hardware specification or architecture.

The use of virtualization and grid middlewares has been previously proposed by Figueiredo et al. (2003), who described the feasibility and benefits of using VMs within a middleware: security, isolation, customization, legacy support and resource control. Specifically for DGC, three different approaches have been published. Santhanam et al. (2005), presented the benefits of employing VMs within Condor (M. Litzkow and Livny 1997) and Xen technology for virtualization (Barham et al. 2003). The main goal of using virtualization is to provide security for running applications, instead of providing a customizable execution environment, as we propose in this work. It is obvious that, thanks to virtualization technology, both middlewares, Condor and BOINC, profit from the intrinsic security feature of virtualization; however we are presenting the virtualization layer as a complete solution to complex applications or systems within a BOINC infrastructure.

The second proposal in DGC was presented by Calder et al. (2005), relying on the Entropia middleware (Chien et al. 2003), a commercial application for DGC. The employed virtualization layer is completely developed by the Entropia team, and is only available for Microsoft

Windows computers, which keeps GNU/Linux and Apple computers from being considered within the described model. The third one, corresponds to the early efforts presented by Lombraña et al. (2007b) for the BOINC DGC middleware (Anderson 2004). The current paper builds on that work, presenting new results obtained thanks to the employment of this approach and a new tool for managing BOINC resources in an institutional environment like a university.

Another solution for using virtualization and DGC middleware, is the proposal presented by the Xtremweb project (Fedak et al. 2001). The Xtremweb middleware was developed using the Java programming language, so the virtualization technology is provided through the Java virtual machine. Despite the usefulness of this approach, the Xtremweb middleware cannot tackle large complex systems where the complexity of the scientific application relies on other external libraries, like the computer vision problem (Olague and Trujillo, 2011) employed for testing the new proposal included in this work.

In conclusion, to the best of our knowledge, all of the above efforts aimed at adopting VMs together with DGC have considered a restricted scope, mainly focusing on security to provide a trusted and secure sandbox facility inside the DGC middlewares or at programming language level. However, our approach adds the benefits of running any scientific application, including EAs, within the BOINC middleware through the use of VMs.

3.1 BOINC: a desktop grid computing technology

BOINC is one of the most employed middlewares in DGC and the leader when referring to volunteer computing. However, BOINC was not the first approach in desktop grid computing. The first approach in volunteer computing was presented by the project Great Internet Mersenne Prime Search (GIMPS)[1].

All the above projects are, more or less, focused on solving a specific problem instead of providing a general tool. Examples of generic tools are Xtremweb (Fedak et al. 2001), Condor (M. Litzkow and Livny, 1997) and BOINC (Anderson 2004). Xtremweb is nowadays employed by hundred of users, while Condor is focused on desktop computers but not on volunteers. On the other hand, BOINC is a *multiplatform* and *open source* middleware that comes from the SETI@home project (Anderson et al. 2002) and provides a general tool for developing new DGC projects based on BOINC. SETI@home has engaged more than a million of volunteers that provide more than two million of computers which gives a computer power of 736 TeraFLOPS. SETI has engaged the largest community and thanks to the new BOINC middleware up to fifty different research projects are running currently, such as the LHC Volunteer Cloud Computing Project[2]. Taking into account all the described DGC approaches, we have chosen BOINC because it is the most employed middleware and it has the largest users community. BOINC technology is made-up of two key elements: the server and the clients. Thus, BOINC has a master-slave architecture, where the server is in charge of:

- *Hosting the scientific project experiments.*
- *Creation and distribution of jobs.*

On the other hand, the BOINC client connects to the server and asks for work (WU). The client downloads the necessary files and starts the computations. Once the results are obtained, the client uploads them to the server. Researchers that want to use this framework to run their experiments have two options for creating a BOINC project:

- *Starting from scratch*. In this case, the researchers will create an application using the supported programming languages by BOINC (C++ and Fortran), and following the guidelines of the framework.
- *Using the wrapper*. The wrapper is a solution for running legacy applications, applications which do not use the BOINC API at all and are statically linked.

Nevertheless, even when this solution fixes most of the porting problems, it is unsufficient when researchers employ more complex environments, like the chosen problem for testing this new approach in this work. In this case, the algorithm is coded using Matlab, a framework that does not provide the source code for linking it with BOINC libraries. Furthermore, the researchers employ different toolboxes and libraries for tackling the problem. As a consequence, researchers will only be able to port their application except if they re-write the whole application from scratch (including external toolboxes and libraries).

The next section presents the virtualization technology and how it is possible to integrate it within BOINC for allowing researchers to run any application within a BOINC system without having to modify the source code.

3.2 Virtualization

Virtualization is a computer paradigm that abstracts the computer's hardware. It basically creates a virtual machine (VM) where it is possible to load an OS and run different applications like in a real computer. The VM is handled by

[1] http://www.mersenne.org.

[2] Boincvm (2010) BoincVM Volunteer Cloud Computing Platform. http://code.google.com/p/boincvm/.

a software called Virtual Machine Monitor (VMM) or hypervisor. The VMM is responsible of virtualizing the underlying hardware. Additionally, the hypervisor avoids that any problem within the VM affects the host machine, the real hardware. The main features of virtualization are:

- *Resource isolation*. Virtualization isolates each VM inside the host machine. In other words, a failure in the guest machine, the VM, does not affect the host machine.
- *Guest OS instantiation*. The instantiation facility permits to create VM images that can be deployed on other hosts that have the same hypervisor.
- *Snapshots* or state serialization (also known as checkpointing (Elnozahy et al. 2002)). The guest machine can be paused and restarted at any point of its execution thanks to this feature.

Examples of VMM are VMware (Nieh and Leonard 2000; Sugerman et al. 2001), VirtualBox (Watson 2008), Xen (Barham et al. 2003) or KVM (Habib 2008). It is obvious that the employment of a virtualization technology adds some overhead, in comparison of running the application directly on the "bare metal". However, reducing the overhead is one of the key points in virtualization technologies (Barham et al. 2003; Sugerman et al. 2001; Habib 2008). For instance, the Xen hypervisor (Barham et al. 2003) has been designed in such a way that its overhead is meaningless in comparison to the real hardware—less than a few percent in CPU, I/O and network (Barham et al. 2003). The same can be applied in the other hypervisors as VMware (Sugerman et al. 2001), KVM (Habib 2008) or VirtualBox (Watson 2008).

In summary, with the employment of the virtualization technology, it is possible to concurrently run several VMs or guests, on a single physical machine, the host. Moreover, those VMs can have different setups including different OSes and platform architectures.

4 Proposal: a virtualization layer within BOINC

Thanks to this new model—BOINC + VMware– researches would be able to run their experiments on parallel environments without modifying the source code of their applications. Additionally, the virtualization layer is transparently "embedded" within the BOINC project, without any substantial differences compared with a standard BOINC project (the BOINC source code has not been modified at all). This model features a parallel approach which is very important in scientific experiments; in particular when experiments must be run repetitively: for example, parameter sweep problems are good candidates for BOINC plus virtualization, as well as stochastic algorithms, where experiments need to be run a large number of times (Olague and Trujillo 2011).

As we have stated previously, BOINC is our choice for extending it with the virtualization technology. As BOINC is multiplatform, supports GNU/Linux, Mac OS X and Microsoft Windows PCs; the VMM hypervisor must support at least the same platforms. Due to this restriction, and taking into account which is the most popular VMM on desktop machines, we have chosen the VMware software as our virtualization hypervisor.

A BOINC experiment provides Work Units (WUs) to clients for computing experiments. Thus, in our approach, a WU will have the following items:

- the *wrapper*, and
- the *VM image*: a set of files needed to boot the VM, which will be the input files for the experiment.

BOINC was not designed with virtualization support, so the integration of VMware and BOINC has some challenging aspects that have to be addressed, these are:

- VMware integration,
- distribution of large files (the VMs),
- installation process of BOINC and VMware, and
- remote BOINC clients management.

VMware is not an open source application, so it is not possible to adapt it to use the BOINC API. The solution for this problem is the employment of the BOINC *wrapper* solution, that allows us to run legacy applications. The wrapper talks directly to the BOINC client, however, the wrapper does not provide all the required flexibility for setting up the VMware environment and launching/managing the VMs. For this reason, a new small program is required and we have called it the *starter*. The aim of the *starter* is to set up the virtual machine: i.e., start, stop or pause the VM when required.

The size of the VM that has to be distributed is also a problem for a BOINC project. The BOINC server network load is going to be affected when a large file has to be simultaneously downloaded by hundreds or thousands of clients. This problem is not specific of our VM proposal, as different research projects will need large files to be analyzed, for instance the BOINC climate change project Climateprediction.net (Allen 1999) is distributing compressed files of several hundreds of MB to run their experiments. For this network issue, Costa et al. (2008) improved the BOINC server and client by adding the well known P2P BitTorrent protocol. Thanks to this solution, the network requirements are reduce by over 90% in comparison with a standard BOINC project for large files, like in our case with VMs.

In this work, we are going to use the BOINC server without BitTorrent support, because we want to test the

standard scenario in one laboratory with a few computers, knowing that if we have to scale, the BitTorrent solution can be employed. Although, the installation of the BOINC client and the VMware hypervisor may be bit problematic (VMware needs administrative privileges to get installed, so it is impossible to deploy it through BOINC), we propose a solution for institutions where software deployment policies allows what is required here: to install both applications independently in the same computer. As described below we also provide software tools to easily manage the computing resources.

The next sections present the *starter* program and the proposed solution, Jarifa, for managing hundreds to thousands of computers from a single and centralized web page.

4.1 The starter

Thanks to the employment of the virtualization technology, a global purpose checkpointing facility can be used by any application, and more importantly, the software can be stopped at any time without requiring any additional knowledge about where it is possible to stop the application for checkpointing. This useful feature is possible thanks to one of the most interesting capabilities of virtualization: snapshots. To harness this feature, we are going to use the *starter* program. This application will launch VMware commands—through the starter—taking snapshots at fixed period intervals, for example each hour or saving the state[3] of the virtual machine when the BOINC client is quitting its execution. Thanks to this approach, if the BOINC client is stopped or the PC is turned off, the scientific application will start again from the last previous saved snapshot or state.

A VMware virtual machine is composed of a set of files containing the definition of the virtual hardware, operating system and applications, see Fig. 1. All these files are treated as standard WU input files. Thus, when the clients have been successfully attached to the server, BOINC clients will request some work to the server. Then, the BOINC server will send the WUs containing the VM image and the starter program for handling the virtual machine. Once the BOINC client has downloaded the VM (a set of files), the starter through the wrapper will set up and launch the VM in VMware. Finally, when the VM has been booted, the scientific application will run within the VM and its results in the VM. At this point, the BOINC client will have to access the obtained results within the VM. VMware provides a solution for accessing a VM folder from the host client. This solution is known as *shared folders*.

[3] Saving the state of a virtual machine is like hibernating a PC.

Fig. 1 A high-level diagram of the the VMware virtual machine

As a security precaution, VMware Player has shared folders disabled by default. For this reason, we have created a script, stored and run inside the VM, that uploads the results via SSH to a remote server, in this case to the BOINC server itself. In this way, we are providing a secure transport layer for uploading the obtained results to the server. Once this process has ended, the VM will be shutdown, and the BOINC client will be able to report a completed task, and ask for more work to the server.

In summary, the whole process in the client (including the installation steps), which are systematically developed by the information technology departments in medium to large institutions, are the following (see Fig. 2):

1. Install the VMware Player in the PC.
2. Install the BOINC client in the PC.
3. Attach the BOINC client to the project.
4. The BOINC client proceeds with the following steps:
   (a) Request work.
   (b) Download the WU files: VMware image, the *starter*, the *wrapper*.
   (c) Launch the *wrapper*.
   (d) The wrapper launches the *starter*.
   (e) The starter launches the VM using the installed client VMware Player:
      i. The VM boots the OS.
      ii. The VM launches the experiment.
      iii. Once the results are obtained, a script uploads the results to a given server (usually, the BOINC server).

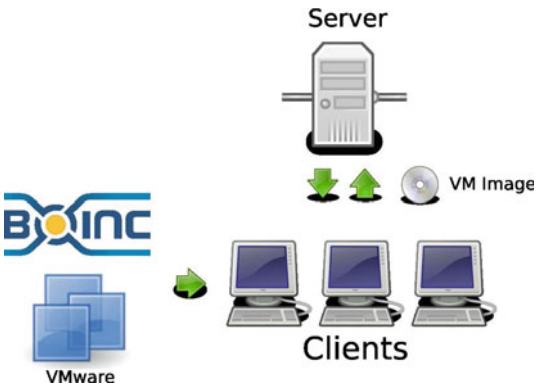


Fig. 2 Conceptual diagram of the BOINC & VMware Enabled Clients working with the BOINC Server

iv. The script powers off the VM, to notify the BOINC client that computations have been finished.

(f) If checkpointing is necessary, take snapshots.

(g) Once the VM has been powered off (step iv), notify the BOINC client that computations have been finished and request new jobs.

Taking into account this new work flow, the researcher will only be responsible of setting up a VM that supports his scientific application or environment. This step is very easy as researchers master their own research environment and the source code is not modified at all, so installing their own software and setting it up on a VM will be a simple step. As researchers can run any possible software (open source or with license restrictions) within the virtual machine, it is the researcher who must follow and respect the software licenses to run the software within this proposal. Usually, this is not the problem given that institutions provide software licenses required by scientists. For this reason, the legal issues that could arise by using BOINC with virtual machines are out of the scope of the objectives of this paper. Thanks to the instantiation feature of virtualization technology, the created VM can be distributed by BOINC to all the clients that are going to collaborate in the project.

Summarizing, to the best of our knowledge, this is the first attempt to provide a complete solution based on a virtualization layer to create customizable execution environments within BOINC.

4.2 Managing hundreds to thousands of BOINC computers

The second part of the solution we are providing is a software tool that allow researchers and institutions to centrally control computing resources. From the point of view of the resources, the institution is building a cluster, with resources distributed in different locations. Hence, the institution should be interested in managing those resources from a centralized web page or application. The standard BOINC model focuses on volunteers that own the computer, so they have the power of choice. In our proposal, the institution has the power of choice, as the resources belong to the institution. Figure 3a shows the standard model of BOINC, where the users have all the power of choice, while Fig. 3b shows the new proposal, where we have "removed" the final user from the classical model, giving the power of choice to the institution or the researcher that manage the BOINC resources.

We present below a new tool designed completely for managing BOINC resources that harnesses its API[4] and does not require the installation of any extra software in the client side. The new administration tool is called Jarifa.

4.2.1 Jarifa

Jarifa is coded using PHP5 and using a MySQL database for storing the information of the BOINC enabled computers. The applications is open source and it is available in http://github.com/teleyinex/jarifa. This new tool performs the following tasks to simplify the management of BOINC resources:

- Managing (create, delete or modify) groups/pools of computers. This feature allows the administrator to create different sets of computers that have different preferences.
- Assigning BOINC projects to computers. Thanks to this feature, the EA researcher can choose which of his research projects can be run in the BOINC resources, assuring that, for example, only his project has full-access to all the computing resources.
- Statistics retrieval. The researcher will like to know how well the system is operating, so the application retrieves statistics from the BOINC computers.

Additionally, as this tool has been designed to be deployed in an institution like a research center or a university, the application is multi-user implementing different roles for privileges. The main roles are:

- *Root*. This user has all the privileges.
- *Allocator*. This profile allows to add/remove BOINC projects to the computing resources, and also set a priority on them.
- *Supplier*. This profile represents users of another institutions that want to collaborate with the main

[4] http://boinc.berkeley.edu/trac/wiki/WebRpc.

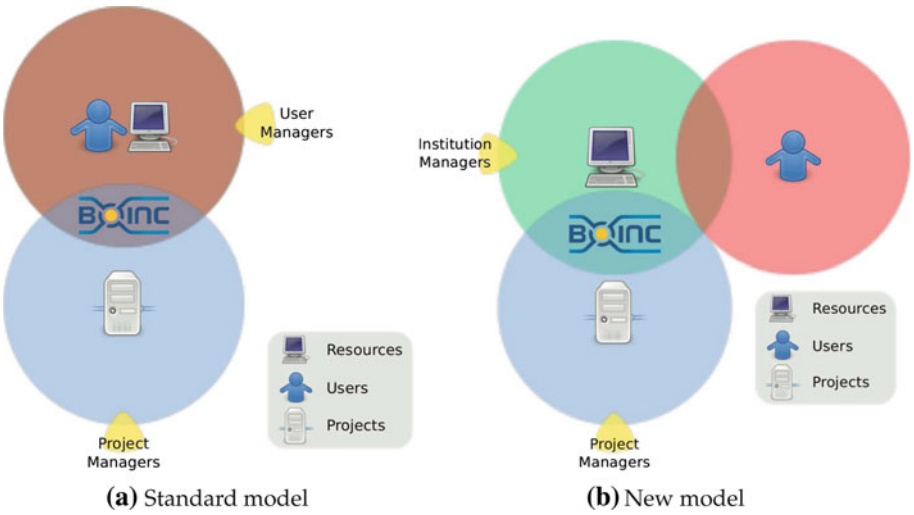


Fig. 3 Conceptual diagrams for BOINC systems: **a** the standard model and **b** the proposed model that provides institutional control

one. For instance, if Jarifa is deployed in an university, all the faculties can be different suppliers providing different computer resources.
– *Volunteer*. This last profile represents the main user of the BOINC community with only one difference, the volunteer trusts the institution, as he cannot chose which projects are going to be run in his resources.

Jarifa does not manage any virtual machine as do other cloud systems. Jarifa only talks to BOINC clients, as these are the BOINC resources that have to be managed. BOINC deploys the virtual machine, start/stop it, etc. so the Jarifa software only has to set up correctly the BOINC preferences. To sum up, the main features of Jarifa are:

– It has been designed following the *Model-View-Controller* architectural pattern (Krasner and Pope 1988).
– Open source (license GNU Affero V3.0[5]) and coded in PHP5.
– The data are stored in a MySQL database.
– It uses the BOINC's Account Manager mechanisms.
– Multi-lingual (supported languages by default: Spanish and English).
– Multi-platform. Supports Microsoft Windows, GNU/Linux and Mac OS X.
– Multi-user. There are four different roles for the users:
  – Supplier.
  – Allocator.
  – Root.
  – Volunteer.
– Geo-localization for volunteers and suppliers on a map.
– HTML and CSS compliant with W3C standards[6].
– Integration with social networks like Twitter or Identi.ca.

4.2.2 *Managing resources with Jarifa*

The main challenge for this software is to be able to manage from tens to thousands of BOINC based computers, depending on the size of the institution. Thus, with this purpose in mind, we tested Jarifa's capabilities toguether with the international project Extremadurathome.org[7] in collaboration with different Spanish institutions. As of March of 2011, the project is managing more than one thousand computers, seven hundred volunteers (we allowed them to participate) and forty seven suppliers (another institutions that join the project, donating their own resources and allowing us to test the software in all its dimensions). Figure 4 shows the projects where the institutions have contributed more based on the BOINC credit (Anderson 2004)[8]—the credit represents how much work has been contributed to a given project. As a consequence of the success of this project, different research centers and institutions have adopted and are using Jarifa for managing their own BOINC resources. For instance, the International Potato Center of Perú, the Centro Informático Científico de Andalucía (CICA) of Spain managing a cluster of 100 nodes, or the Consejería de Educación de la Junta de Extremadura, managing more than fifty thousand computers.

In summary, Jarifa allows to manage BOINC resources from a centralized web page, simplifying the management of a desktop grid system based on BOINC. Thanks to Jarifa and the proposed solution of the virtualization layer within

[5] Foundation FS (2009) Gnu affero general public license. http://www.fsf.org/licensing/licenses/agpl-3.0.html.
[6] The w3c markup validation service. http://validator.w3.org/.
[7] Extremadurathomeorg (2010) http://www.extremadurathome.org.
[8] These data can be consulted on-line and in real time in the following URL http://boincstats.com and http://extremadurathome.org.

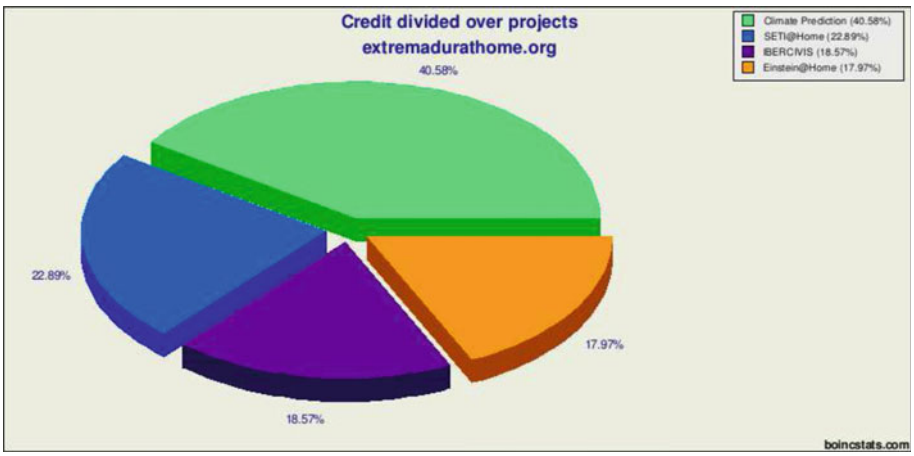


Fig. 4 Pie chart of the percentage of users from the Extremadurathome.org community that chose each BOINC project

Table 1 Hardware specifications for BOINC server and clients

	CPU	RAM	HD
Server	Intel P4	1 GB	300 GB
Client	AMD Sempron 3000	256 MB	40 GB

the BOINC middleware, any EA researcher will be able to run his experiments in the proposed custom execution environment without having to modify the source code of his applications, and manage the BOINC resources from a centralized web page (Table 1).

5 Distributed genetic programming with BOINC + virtualization

In order to illustrate the benefits of the proposal, we chose a very complex computer vision problem requiring a non-statically linked software using GP (Koza 1992) as the underlying technique. GP is an automated method to create a working computer program for a given problem using tree-like structures as the representation of the individuals. More details on the problem and application of GP to solve the problem are summarized below, and can be also found at (Trujillo and Olague 2008; Olague and Trujillo 2011). The proposed problem provides a very interesting real-world test for our DGC framework for the following reasons:

1. It is coded in Matlab and it is not trivial to port all of the algorithms from Matlab to C++ or Fortran. Moreover, the algorithm has several dependencies with Matlab toolboxes.
2. The algorithm is very time-consuming, which makes it a good candidate for parallel environments such as the BOINC client infrastructure. In our setup, every client will run the experiment using a single VM.

Although hundreds of computers were available at our institution, twenty of them were selected for this specific experiment. Jarifa allows to specifically select pools of computers—within the whole institution—so that different pools can contribute to different projects. Half of the desktops were running Microsoft Windows and the other half GNU/Linux. A BOINC server was also set up for hosting the experiments and distribute them to the BOINC-VMware clients (see Tab. 1 for server and client hardware specifications).

For creating the VM image we used the freeware VMware server. The chosen OS for the VM image was a GNU/Linux standard distribution: a Debian OS. As the selected computer vision problem requires a large amount of computing time to produce a single possible solution—more or less 24 h on a monoprocessor system (Trujillo and Olague 2008), we set up the VM so that each client could generate a maximum of five solutions, which is more than the expected best case scenario. Due that the application last in average 24 h to produce a single solution, the employment of the virtualization snapshots facility is useful, as it is not possible to lose 24 h of work. Saving and restoring a VM takes less than 2 min, depending on other I/O activities on the computer it could be greater, it is worthy to save the state of the virtual machine, when the BOINC client is going to be stopped. We used a minimal installation for Matlab and Debian, using only indispensable toolboxes for the former in order to reduce the total size of the VM image. The final size of the VM was of 1.4 GB. A fixed period of 48 h was chosen for running the experiment, during which the BOINC clients were never shut down and all computing resources were assigned to the computer vision algorithm. When a client found a solution, a script uploaded the result to a FTP server via a SSH connection in order to ensure confidentiality. Also, when the experiments began, we encountered a problematic situation that is common in real-world DGC environments: the GNU/Linux computers underwent a series of

technical interventions by maintenance staff (installation and upgrade of available software on all platforms: Ms Windows and GNU/Linux) that adversely affected the on-going experimentation. Therefore, the BOINC project lost CPU power, and only the Microsoft Windows computers were 100% operative during the 48 h period. These types of events represent a typical non-optimal situation for DGC systems, where resources are sometimes lost during on-line experimentation (host churn) (Kondo et al. 2007). Thus, the experimental setup constitutes a perfect example of a real-world deployment that allows us to effectively evaluate the usefulness of our approach.

In what follows, we briefly describe the scientific problem that is used in this experimental setup, and evaluate the results that were produced from two perspectives: (1) the performance of the proposed BOINC infrastructure; and (2) results relative to the problem itself.

5.1 Real-world problem: interest point detection with GP

Currently, a large number of computer vision systems employ a local approach to feature extraction and image description. Such systems focus on small and highly invariant point features called interest points, which can be easily and efficiently detected. Interest points can be described as salient image pixels that are unique and distinctive; i.e., they are quantitatively and qualitatively different from other image points, and normally interest points only represent a small fraction of the total number of image pixels.

In (Trujillo and Olague 2008; Olague and Trujillo 2011) a method was proposed which automatically synthesizes image operators that detect interest points. The design of an interest point detector was posed as an optimization problem and solved with genetic programming (GP), a form of evolutionary computation (Koza 1992). The fitness function, that measures the performance of each solution that the GP generates, considers the stability of the detector measured with the repeatability rate, and the amount of dispersion that the set of detected points exhibit over the image. The repeatability rate is computed between two images taken from different viewpoints; thus, one is the base image and the other is its transformed counterpart. After detecting interest points on the base image it is possible to inspect if the same scene features are marked with interest points on the transformed image. Hence, the repeatability rate can vary from zero, meaning that no points are repeated and detection is completely unstable with respect to the transformation, to one hundred, which means that detection is completely stable and invariant. Experimental results have confirmed that operators produced in this way are indeed competitive with state-of-the-art detectors that are widely used in computer vision applications.

The evolutionary computation paradigm consists on the development of computer algorithms that base their core functionality on the basic principles of Neo-Darwinian evolution. Genetic programming (GP) is arguably the most advanced and complex technique used in evolutionary computation, it is a generalization of the better-known, and more widely used, genetic algorithms (Koza 1992). In GP, each individual solution is represented using tree structures because they are able to express simple computer programs, functions, and mathematical operators. Hence, the main particularities of a GP algorithm, compared with other evolutionary and population based optimization methods, are those related to the manner in which trees are created, combined and randomly modified. In practice, when applying GP to a specific problem the most important task is to define the set F of functions that can be used as tree nodes, and the set T of terminal elements that can appear as tree leaves. These sets define the search space where the GP can generate and test individual programs.

Although the results published in (Trujillo and Olague 2008; Olague and Trujillo 2011) have been encouraging, a severe bottleneck for the GP algorithm is the time that is required to complete a single successful run, which can sometimes take as many as 24 h. Therefore, developing a deeper understanding of the GP algorithm can become difficult just because obtaining the necessary experimental data requires so much time. The framework proposed in this work allows us to execute several experiments with the GP algorithm in a parallel manner, and test the following aspects:

1. Increase the size of the training sequence used during the evaluation of each solution produced by the GP algorithm.
2. Reduce the size of the terminal and function sets, T and F, by eliminating those that were originally included in the experimentation carried out in (Trujillo and Olague 2008; Olague and Trujillo 2011) but that were rarely used as part of the best solutions found.

In order to carry out the tests described above, the GP algorithm was setup with the configuration described in Table 2.

5.2 Results

This section summarizes the results produced by our experimental setup, we discuss the performance of the proposed BOINC infrastructure, and give an overview of the solutions generated for the computer vision problem we addressed.

Table 2 Experimental setup of the GP algorithm that was used with the BOINC/VMware framework

Parameter	Value
Population size	75
Generations per run	75
Function set	$F_A = \{+, -, \lvert I_{out}\rvert, *, \div, I_{out}^2, \sqrt{}, log_2\} \bigcup \{\frac{\delta}{\delta x} G_{\sigma_D}, \frac{\delta}{\delta y} G_{\sigma_D}, G_{\sigma=1}, G_{\sigma=2}\}$
Terminal set	$T_A = \{L_x, L_{xx}, L_{xy}, L_{yy}, L_y\}$

5.2.1 The BOINC DGC environment

During the 48 h period used for experimentation the available computing resources produced 12 solutions. Each execution required a different amount of computing time due to the stochastic nature of the GP algorithm. The total time employed on obtaining the solutions was of 215 h, while the average per solution was of 18 h. Thanks to the used parallel approach, we saved 9 days if the same solutions have to be computed by a single computer. Thus, the speedup relies on the amount of clients that we can attach to the BOINC server. As this proposal is presented for institutions, the speed up will depend on the available number of desktop computers that belong to the institution.

Finally, we must emphasize that we did not need to modify the original source code in any way in order to achieve the reported speed-up for the GP algorithm, nor did we compile different versions of the algorithm to account for the different platforms that were used. Therefore, the computing model presented here is very easy to scale because the computing resources can be increased by simply adding new desktops clients with BOINC and VMware installed, a process that is both simple and relatively fast.

5.2.2 The GP algorithm for interest point detection

The GP algorithm generated a total of 12 image operators for interest point detection, one in each run. Figure 5 presents the convergence tendencies of the GP algorithm using the average and standard deviation obtained from the 12 runs. The plot shows the fitness value of the best solution and the average population fitness obtained at each generation. However, a similar solution was basically obtained in eleven of those, a Laplacian operation. It appears that given the limited search space, compared with the one used in (Trujillo and Olague 2008), the GP converges to the same local maxima. Nevertheless, the one solution that was unique did provide an interesting operator that computes an interest measure using the relative intensity of each image pixel, given by, Eq. 1:

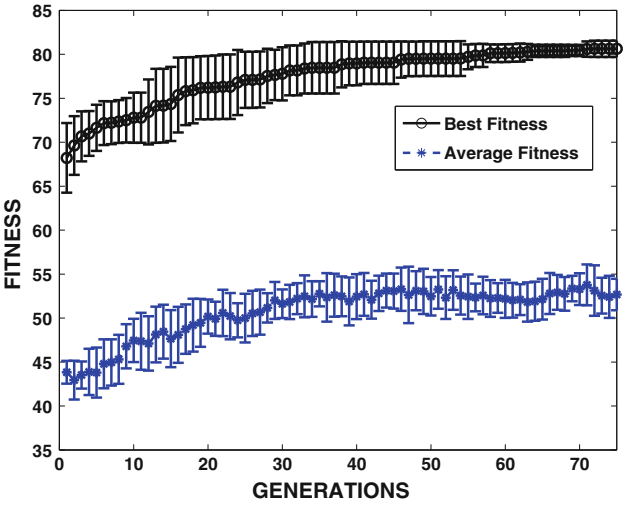


Fig. 5 Convergence of the GP algorithm, showing the best fitness at each iteration and the average population fitness. The plot shows the average over the 12 different runs and the corresponding standard deviation

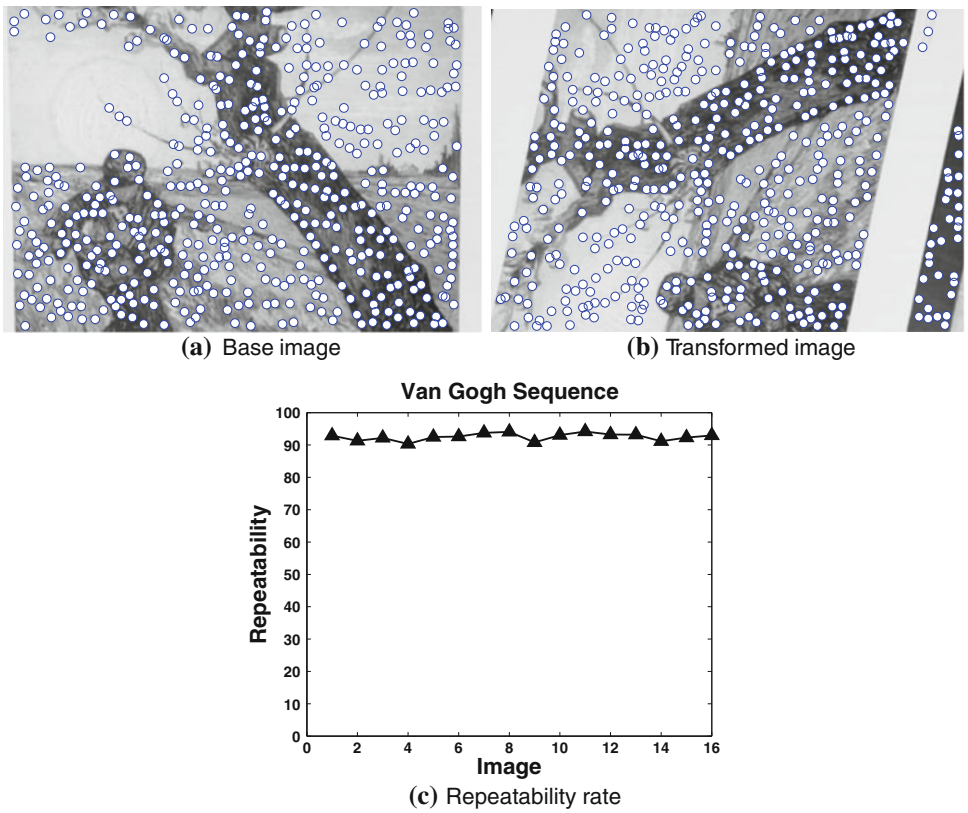
$$K(\mathbf{x}) = G_{\sigma=1} * \sqrt{G_{\sigma=2} * \frac{1}{I(\mathbf{x})}}. \quad (1)$$

where $\mathbf{x}$ is an image pixel, G_σ are Gaussian filters with a blur factor of σ, and $I(\mathbf{x})$ is the intensity value of pixel $\mathbf{x}$. This operator will enhance pixels that are darker than the weighted average intensity of surrounding pixels computed with a Gaussian mask.

In Fig. 6a and b we present sample interest points that were detected using the above operator on two images from the Van Gogh image sequence (Trujillo and Olague 2008; Olague and Trujillo 2011), which was also used as a training set. Finally, Fig. 6c shows the repeatability score of this detector computed using the base image of the Van Gogh sequence, Fig. 6a, with respect to sixteen progressively transformed images similar to the one shown in Fig. 6b. It is important to point out that the performance of this detector is comparable to that achieved by state-of-the-art detectors used in computer vision.

5.3 Shortening time for evolutionary computation and scientific applications

Although the literature has usually presented results concerning the speedup obtained when parallel versions of EAs are deployed [6], to the best of our knowledge, no information on the time devoted to modify a previous sequential code or the time required for developing a new parallel program has been published. Nevertheless, development time is crucial for hardware and software industry, as well as for researchers (Asanovic et al. 2009).



(a) Base image **(b)** Transformed image

(c) Repeatability rate

Fig. 6 Performance of the operator produced by the GP algorithm for the Van Gogh sequence. **a** Interest points detected on the base image. **b** Interest points computed on a transformed image. **c** The repeatability rate computed for every image in the sequence

EA researchers have typically tried to develop parallel version of their tools by directly modifying sequential code that is already available, or developing new tools instead. While the development cost of a new software application can be estimated (Vahid Khatibi, 2010), this paper considers the *easy* way of experimenting with Parallel or Distributed Evolutionary Algorithms: modifying a standard sequential version of a given EA tool. In any case, the first option is available, and although no explicit information on development times for distributed EAs are available, they can be inferred by consulting the corresponding literature. Consider for instance the 3-year term European funded project DREAM, whose main result was a Distributed Evolutionary Algorithms framework. Attending at the information provided by the project, at least 2 years where required for this new software to be developed and released (Arenas et al. 2003).

The problem is that researchers usually want to run experiments on specific infrastructures they have access to and available tools usually don't fit them: They face the need for parallel versions of available code. Therefore, we consider here the effort required to adapt an available tool to a specific parallel/distributed model. Data collected from experiments developed with standard EA tools are provided in Table 3, which illustrates the cost associated with code parallelization; several models are considered:

- Using a communication library so that the sequential version is improved to run on parallel and distributed infrastructure. Some of the best known libraries have been considered by previous researchers, such as PVM or MPI (Squyres 2005).
- Using a framework allowing grid deployment of a previous parallel tool. The case considered relies on BOINC.
- Using GPU technology and special libraries for code development. In this case researchers usually develop new code, given the differences among standard sequential programming models and General Processing on GPUs (Owens et al. 2007).

In any of the above cases, the learning curve for the software developer must be considered along with the cost of code development. Table 3 includes technologies, the EA tool used in the study, the time required for code modification—provided by the researchers-, which includes the learning time for the programmer/researcher, and some of the references where results obtained with the new parallel/distributed implementation were published.

Table 3 A comparison of development costs when porting a sequential EA to a parallel system

Technology	Sequential EA tool	Development time (provided by developers)	Published results
PVM	GPC++	1 month	Fernández et al. (1999)
MPI	GPC++	1 week–1 month	Fernández et al. (2000)
BOINC	LilGP	1 month	Chavez et al. (2007)
CUDA	New software	>1 month	Contreras et al. (2012)

Although these are example cases, the development time can be broadly considered as a standard estimation for each of the technologies involved. As we may notice, the average approximated time that researchers have provided (in some cases some of our previous research) is between 1 week (improving to MPI after a PVM code) and 1 month. In the case of MPI, developers were those who had previously released the PVM version of GPC, so the learning curve for the sequential tool, communication libraries, etc., are hidden when the same programmer is applied to the task. If a new programmer enters then the same time is required.

The main conclusion is clear: for a new researcher trying to run a parallel version of an EA an average of a month is required before the experiment is performed. Therefore, if we consider the total time for obtaining a series of results we should add this extra time. Given that the literature that deals with "hard problems" usually considers a time to solution above several days, we see that the time before the experiment is run could be above 90% of the total time required to obtain solutions. Of course, this time is required for the first set of experiments, and then a series of experiments can be run without extra time. But we are considering here precisely the setup time, and trying to reduce this, so a new researcher to the field can begin launching experiments as soon as possible. Once experiments are running, we still have the benefit of the speed-up provided by the parallel infrastructure. Therefore we reach our initial question: is it possible to offer a technology allowing researchers to automatically run parallel experiments avoiding learning curves + development time? We are thus effectively removing more than 90% of time-to-results.

6 Summary and conclusions

Evolutionary computation is a powerful and robust search technique that can be easily adapted for a wide variety of application domains. However, an evolutionary search also comes with a cost, in many real-world scenarios it can be computationally expensive, requiring very long execution times. The limitation is made even more prominent by the fact that a large amount of experimental runs are usually required to gain a deeper understanding of the underlying dynamics of an evolutionary search. This is particularly true for complex search spaces with ill-defined fitness landscapes, such as in Genetic Programming.

Therefore, an important research problem is developing distributed or parallel systems that facilitate the massive execution of experimental tests. This paper presents an extended model based on virtualization and BOINC based on DGC platforms, that provides transparent customizable execution environments. This model allows researchers to run their experiments and applications without any modifications, a strategy with a very steep learning curve, where dependencies produced by differences in programming languages, libraries, hardware platforms or OSes are thus removed. Therefore, every available desktop PC can provide computing power for an application, regardless of its hardware and software features.

The system has been fully tested with a real computer vision application of Genetic Programming, with the following results of interest: (i) a complex scientific environment (Matlab and different toolboxes, together with graphic libraries and scripts) has been deployed and run on a BOINC based DGC environment with heterogeneous PCs (ii) the computing power provided by the system has saved long computing time (iii) results obtained by the specific application are comparable to the state of the art in computer vision systems. Although the virtualization layer introduces some overhead to the system, its impact is negligible when compared with the benefit of running any application on a BOINC infrastructure without any adaptation or change.

Finally, we have also developed a new web tool for managing BOINC resources from a unique and centralized web page, simplifying the management of a desktop grid system based on BOINC. The application has been successfully tested in different real projects, and as a consequence different international research centers are using it. In summary, this paper presents a full solution for running any scientific application—including EA experiments—within a desktop grid by using virtualization technologies and a centralized web page, thanks to Jarifa, that simplifies the management of all available BOINC resources.

Future work on this topic will focus on testing massive deployments of Distributed Evolutionary Algorithms, developing detailed studies of new load-balancing techniques that take into account the special nature and dynamics of an evolutionary process. For instance, a

particularly interesting case is GP and other variable length representations that produce a high degree of heterogeneity within the evolving population. Moreover, the inherent churn phenomenon must be studied with regard to the overall volunteer computing paradigm.

Acknowledgements This research was funded by the Spanish Ministry of Science and Innovation under project ANYSELF (TIN2011-28627-C04), FEDER and Gobierno de Extremadura project GR10029, and IdenTIC. Additional funding was also provided by CONACyT, México, through Project 155045—"Evolución de Cerebros Artificiales en Visión por Computadora". Finally, thanks are given to the support provided by the Departamento en Ingeniería Eléctrica y Electrónica from the Instituto Tecnológico de Tijuana.

References

Allen M (1999) Do it yourself climate prediction. Nature 401 (6754):642–642

Anderson D (2004) Boinc: a system for public-resource computing and storage. In: Proceedings of the fifth IEEE/ACM international workshop on grid computing, pp 4–10

Anderson DP, Cobb J, Korpela E, Lebofsky M, Werthimer D (2002) Seti@home: an experiment in public-resource computing. Commun ACM 45(11):56–61

Arenas M, Collet P, Eiben A, Jelasity M, Merelo J, Paechter B, Preuß M, Schoenauer M (2003) A framework for distributed evolutionary algorithms. Lecture Notes in Computer Science, pp 665–675

Asanovic K, Bodik R, Demmel J, Keaveny T, Keutzer K, Kubiatowicz J, Morgan N, Patterson D, Sen K, Wawrzynek J, Wessel D, Yelick K (2009) A view of the parallel computing landscape. Commun ACM 52(10):56–67

Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A (2003) Xen and the art of virtualization. Proceedings of the nineteenth ACM symposium on operating systems principles, pp 164–177

Calder B, Chien AA, Wang J, Yang D (2005) The entropia virtual machine for desktop grids. In: VEE '05: proceedings of the 1st ACM/USENIX international conference on virtual execution environments, pp 186–196

Chavez F, Guisado JL, Lombrana D, Fernández F (2007) Una herramienta de programacion genética paralela que aprovecha recursos publicos de computacion. In: MAEB'2007, V Congreso Espa nol sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, Puerto de la Cruz, Spain

Chien A, Calder B, Elbert S, Bhatia K (2003) Entropia: architecture and performance of an enterprise desktop grid system. J Parallel Distrib Comput 63:597–610

Contreras I, Jiang Y, Hidalgo JI, Núñez-Letamendia L (2012) Using a gpu-cpu architecture to speed up a ga-based real-time system for trading the stock market. Soft Comput 16(2):203–215

Costa F, Silva L, Kelley I, Fedak G (2008) Optimizing the data distribution layer of boinc with bittorrent. 2008 IEEE international symposium on parallel and distributed processing, 2008 IPDPS, pp 1–8

De Jong K (2001) Evolutionary computation: a unified approach. The MIT Press

Desell T, Anderson DP, Magdon-Ismail M, Newberg H, Szymanski B, Varela CA (2010) An analysis of massively distributed evolutionary algorithms. In: Proceedings of the 2010 international conference on evolutionary computation (IEEE CEC 2010), Barcelona, Spain, pp 1–8

Elnozahy E, Alvisi L, Wang Y, Johnson D (2002) A survey of rollback-recovery protocols in message-passing systems. ACM Comput Surv (CSUR) 34(3):375–408

Fedak G, Germain C, Neri V, Cappello F (2001) XtremWeb: a generic global computing system. Proceedings of the IEEE international symposium on cluster computing and the grid (CCGRID'01)

Fernández F, Sanchez JM, Tomassini M, Gomez JA (1999) A parallel genetic programming tool based on PVM. In: Dongarra J, Luque E, Margalef T (eds) Lecture Notes in Computer Science, vol 1697, pp 241–248

Fernández F, Tomassini M, Vanneschi L, Bucher L (2000) A distributed computing environment for genetic programming using MPI. In: Dongarra JJ, Kacsuk P, Podhorszki N (eds) Lecture Notes in Computer Science, vol 1908, pp 322–329

Fernández F, Tomassini M, Vanneschi L (2003) An empirical study of multipopulation genetic programming. Genet Progr Evol Mach 4(1):21–51

Figueiredo R, Dinda P, Fortes J (2003) A case for grid computing on virtual machines. In: International conference on distributed computing systems, IEEE Computer Society; 1999, vol 23, pp 550–559

Habib I (2008) Virtualization with kvm. Linux J 2008(166):8

Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor

Kesselman C, Foster I (1999) The grid: blueprint for a new computing infrastructure. Morgan Kaufmann

Kondo D, Fedak G, Cappello F, Chien AA, Casanova H (2007) Characterizing resource availability in enterprise desktop grids. Future Gener Comput Syst 23(7):888–903

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge

Krasner G, Pope S (1988) A description of the model-view-controller user interface paradigm in the smalltalk-80 system. J Object Oriented Progr 1(3):26–49

Laredo JLJ, González DLn, De Vega FF, Arenas MG, Guervós JJM (2011) A peer-to-peer approach to genetic programming. In: Proceedings of the 14th European conference on Genetic programming, EuroGP'11, pp 108–117

Lombraña D, Fernández F, Segal B, Grey F (2007a) Enabling desktop pcs with virtualization for grid computing. In: 1st Ibergrid 2007, Santiago de Compostela, Spain, vol 1, pp 160–171

Lombraña D, Fernández F, Trujillo L, Olague G, Segal B (2007b) Customizable execution environments with virtual desktop grid computing. In: 19th parallel and distributed computing and systems, PDCS, Massachusets, USA, vol 1, pp 7–12

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

Melab N, Cahon S, Talbi E-G (2006) Grid computing for parallel bioinspired algorithms. J. Parallel Distrib Comput 66(8):1052–1061

Merelo Guervós JJ, Castillo PA, Alba E (2010) Algorithm: evolutionary, a flexible perl module for evolutionary computation. Soft Comput 14(10):1091–1109

Nebro AJ, Luque G, Luna F, Alba E (2008) DNA fragment assembly using a grid-based genetic algorithm. Compu OR 35(9):2776–2790

Nieh J, Leonard OC (2000) Examining VMware. j-DDJ 25(8):70, 72–74, 76

Olague G, Trujillo L (2011) Evolutionary-computer-assisted design of image operators that detect interest points using genetic programming. Image Vision Comput 29(7):484–498

Owens JD, Luebke D, Govindaraju N, Harris M, Krüger J, Lefohn AE, Purcell TJ (2007) A survey of general-purpose computation on graphics hardware. Comput Graph Forum 26(1):80–113

Poli R, Langdon WB, Clerc M, Stephens CR (2007) Continuous optimisation theory made easy? Finite-element models of

evolutionary strategies, genetic algorithms and particle swarm optimizers. In: Proceedings of the 9th international conference on foundations of genetic algorithms, FOGA'07, pp 165–193

Pressman R (2004) Software engineering: a practitioner's approach, 6th edn. McGraw-Hill

Santhanam S, Elango P, Arpaci-Dusseau A, Livny M (2005) Deploying virtual machines as sandboxes for the grid. In: Second workshop on real, large distributed systems (WORLDS 2005), San Francisco, CA

Squyres JM (2005) The spawn of MPI. ClusterWorld Magazine, MPI Mechanic Column 3(2):40–43

Sugerman J, Venkitachalam G, Lim B (2001) Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. Proceedings of the 2001 USENIX annual technical conference. Boston, Massachusetts, 15 pp, June 25th–30th, USA

Trujillo L, Olague G (2008) Automated design of image operators that detect interest points. Evol Comput 16(4):483–507

Vahid Khatibi DNAJ (2010) Software cost estimation methods: a review. J Emerg Trends Comput Inform Sci 2(1):21–29

Watson J (2008) Virtualbox: bits and bytes masquerading as machines. Linux J 2008(166):1