ISSN 22773061

Using SOAP and REST web services as communication protocol for distributed evolutionary computation

P.A. Castillo, P. García-Sánchez, M.G. Arenas, A.M. Mora, G. Romero, J.J. Merelo
Department of Computer Architecture and Computer Technology. CITIC-UGR.
University of Granada, Granada, Spain
pacv@ugr.es

ABSTRACT

Designing heterogeneous distributed systems requires of the use of tools that facilitate the deployment and the interaction between platforms. In this paper we propose using Simple Object Access Protocol (SOAP) and REpresentational State Transfer (REST), two main approaches for creating applications based on distributed services, for distributed computation. Our aim is to demonstrate how they could be used to develop evolution- ary computation systems on heterogeneous platforms, taking advantage of their ability to deal with heterogeneous infrastructures and environments, and giving support for parallel implementations with a high platform flexibility. Both approaches are different and present some advantages and disadvantages for interfacing to web services: SOAP is conceptually more difficult (has a steeper learning curve) and more ”heavy-weight” than REST, although it lacks of standards support for security. The results obtained on different experiments have shown that both SOAP and REST can be used as communication protocol for distributed evolutionary computation. Results obtained are comparable, however for large amounts of data (big messages), REST communications take longer than SOAP communications.

Indexing terms/Keywords

Web services technologies; Simple Object Access Protocol; REpresentational State Transfer; Distributed evolutionary computation.

Academic Discipline And Sub-Disciplines

Computer Science; Soft Computing

SUBJECT CLASSIFICATION

Computer Science Subject Classification

TYPE (METHOD/APPROACH)

Qualitative Research; Experimental

ISSN 22773061

INTRODUCTION

Nowadays, the design of distributed systems should ensure interaction among very different platforms. Using web services technologies for distributed computation might provide a common interface that can be called from almost any programming language to develop evolutionary computation systems on heterogeneous platforms. Services are computational resources that can be utilized and organized through the Service Oriented Architecture (SOA) [67] paradigm. In order to use this paradigm, the service providers publish the descriptions (or interfaces) of the services they offer in a service registry, so the service requesters can discover them and bind to the correspondant service provider.

An important SOA capability is that it is not focused on a specific implementation, but offers a set of guidelines to help the developers [11]. The order of service execution is not necessarily static, because the services are designed to be used in a non-established and configurable order. Furthermore, services can be dynamically discovered and used (while in Object Oriented Programming must be previously known and can not change during execution), being also one of the most important capabilities the (optional) distribution in a network.

The web services are the key point of integration for different applications be- longing to different platforms/languages/systems since they are based in a set of standards that make them independent of the underlying technologies used for providing them. Although there are several technologies for developing web services (SOAP, REST or XMLRPC among others [28]), nowadays the more popular approaches are SOAP [16] and REST [40,41,84].

SOAP is the traditional, standards-based approach, but the majority of the web services with public API offer REST interfaces, while some of them offer both REST and SOAP and very few offer just SOAP. All of the major web services providers use REST: Twitter, Yahoo's, Flickr, del.icio.us, pubsub, bloglines, technorati, and several others. Both eBay and Amazon have web services for both REST and SOAP.

On the other hand, SOAP web services are used in lots of enterprise software as well; for example, Google implements their web services using SOAP, with the exception of Blogger, which uses XML-RPC, an early and simpler pre-standard of SOAP.

The philosophies of SOAP and RESTful web services are very different. Strictly, SOAP is a protocol for distributed computing, whereas REST adheres much more closely to a web-based design. SOAP requires a greater implementation and understanding effort of the client side, while REST based APIs focus these efforts on the server side.

It is important to note that one of the advantages of SOAP is the use of the ”generic” transport. While REST today uses HTTP/HTTPS, SOAP can use almost any transport to send the request. However, one perceived disadvantage is the use of XML [73] because of its verbosity, and the time necessary to parse it.

Whatever the technology used to deploy web services, they provide several advantages [46], like language independence and distribution mechanisms; it also increases the interoperability between different software elements (for example, it is possible to add communication libraries without modifying existing code), and facilitates code distribution (it is not required the use of a concrete implementation or library) among geographically distributed work teams.

In this paper, a high-level comparison of the SOAP and REST approaches is made. We propose using them for distributed evolutionary computation, taking advantage of their ability to deal with heterogeneous infrastructures and environments. We intend to innovate in terms of the implementation, because implementation matters [62], as new technologies lead to new ways of implementing the algorithms, they allow new developers to work on new methods and using many computers on distributed experiments.

In order to determine the efficiency of these two interfacing approaches, we have performed four experiments in which both a SOAP and REST implementations are evaluated, taking a ”toy problem” as proof of concept, a function optimization problem, and a complex artificial neural network design and optimization problem. As first experiment a client-server model is implemented, in which the server process runs on a machine and the client processes send and receive text strings. Next, a master-slave based genetic algorithm (GA) is implemented, running on the master process the GA and the fitness evaluation on the slave processes. Then, based on the previous one, a real-coded EA to solve function optimization problems has been tested on three well known functions taken from the CEC2005 benchmark. Finally, as fourth experiment, we implement a distributed evolutionary algorithm (EA) to solve a costly problem: tuning learning parameters and to set the initial weights and hidden layer size of a multilayer perceptron (MLP), based on an EA and Quick Propagation [38] (QP) to solve classification problems.

This work continues with our previous research in service oriented algorithms [48,25], or evolutionary optimisation of MLP (G-Prop method) presented in [22,23].

The main idea of this paper is to study what are the possibilities of this setup as a flexible meta-computer by implementing an EA using web services, and then measuring the speedup when several computers are used to solve a costly classification problem, so that it takes time enough to get some improvement from parallelization. We will only measure how running time scales when new (heterogeneous) nodes are added to the system, being the main objective to test if this kind of system is suitable for scientific computation.

This paper is structured as follows: In section 2 a comprehensive description of SOAP and REST technologies are provided. Section 3 presents a review of the approaches found in the bibliography to describe and implement parallel and distributed EA. Section 4 describes the experiments in detail. In concrete, the client-server and master-slave models implemented for testing are described, as well as the experimental configuration, and the methodology considered in the

study; finally, the results obtained are shown. Last section (Section 5), throws some conclusions and presents the proposed future work.

2. COMPARING SOAP AND REST SERVICES

This section presents a comprehensive description of SOAP (Section 2.1) and REST (Section 2.2) technologies, followed by a comparison of SOAP and REST programming models (Section 2.3).

2.1. SOAP: Simple Object Access Protocol

SOAP is a standard protocol proposed by the W3C [16] to interface web which extends the XML remote procedure call (XML-RPC) standard. SOAP is a complete and mature protocol that allows to perform remote method calls to distributed routines (services) based on an XML interface.

SOAP clients can access to objects and methods that are residing in remote servers, using a standard mechanism that makes transparent the details of implementation, such as the programming language of the routines, the operating system or the platform used by the provider of the service. At the moment, there exist complete implementations of SOAP for Perl, Java, Python, C++ and most modern languages (http://www.soaprpc.com/software/). In opposite to other remote procedure call methods, such as RMI (*remote method invocation*, used by the Java language) or XML-RPC, SOAP has two main advantages: it can be used with any programming language, and it can use any type of transport (HTTP, SHTTP, TCP, SMTP, POP and other protocols).

SOAP messages are sent using XML. The interfaces of the methods that can be accessed are specified by a Web Services Description Language (WSDL) [4]. The WSDL of a web service consists in an XML description of its interface, i.e., it is a file that describes the name of the methods, the parameters (number and type) and the type of response.

In this way, SOAP constitutes a high level protocol, making easy the task of distributing objects among different servers, and avoiding the difficulties derived of defining the message formats, nor the explicit call to remote servers.

One of the advantages of using web services, is that the aplication stack is growing with the WS-Extensions (http://docs.oasis-open.org/ws-sx/). That is, the basic specifications of Web Services (such as SOAP) can be extended with transactions, security or messaging, for example. The most used are:

- WS-Adressing (authentication).
- WS-Security, WS-SecureConversation and WS-Trust (authorization and secure messaging).
- WS-Policy and WS-Metadata Exchange (policy mechanisms for interactions).
- WS-Reliable Messaging and WS-Transaction (add-on mechanisms for the communication channel).

Also, functional extensions, such as WSRF [45], allows the discovery, inspection and interaction with stateful resources in standard and interoperable ways.

Several studies about e-science taking advantage of web services can be found in bibliography [65,30,58,69].

2.2. REST: Representational State Transfer

Representational State Transfer (REST) is an alternative method for building web services. This technology was proposed and defined by Roy Fielding [40,41].

In a REST-style architecture, a client sends requests to the server who process them and return responses to the client. Requests and responses represent resources that can be addressed by an Uniform resource identifier (URI). Usually, resources are documents or programs the client need to access to.

REST usually works on the HTTP protocol, however it can be based on other protocols that provide the appropriate mechanisms to send requests and return responses.

In a REST environment, while servers are not concerned with the client state, clients only take care about their own state and how to address resources on the server using URIs. Moreover, clients can cache responses to improve performance. As the client-server communication is stateless, servers are simpler and more scalable. Taking this into account, if the REST interface is not altered, servers and clients can be modified independently. Finally, servers can customize the functionality of the clients by sending logic (code) to them that can be executed.

REST web services are simple and lightweight (as no extra XML markup is needed), their message format is readable by humans, they are easy to build, and finally, developments achieve a high performance [28].

2.3. Comparing SOAP and REST programming models

The main differentiating factor is that SOAP Services tend to be operation-based, while REST services are resource-based. That means clients call methods on a SOAP service, while on a REST service they send HTTP requests to an URI and expect to get some resource in return [28].

From an architectural perspective what this means is that service operations are inherently more constrained than resources that are freely accessible.

To make data available to clients that focus on the speed (especially clients that may not understand or care about SOAP), a REST-based service is more efficient [28].

For the sake of comparison, Table 1 shows the main strengths and weaknesses for both SOAP and REST (adapted from http://ajaxonomy.com/2008/xml/web-services-part-1-soap-vs-rest).

Table 1. Results obtained on the third experiment (master-slave implementations). The best results are highlighted in order to ease interpretation (asterisks indicate significant differences regarding the other results).

SOAP	
Strengths (pros)	Weaknesses (cons)
+ Handle distributed computing environments + Built-in error handling + Extensibility + Language, platform, and transport agnostic + Prevailing standard for web services + Security + Support from other standards (WSDL, WS-*) + Protocol at different layers (transport, application...) + Requirements that covers: interoperability, synchronization, security, state or contract-based transactions + Extensible with WS-* + Application logic as a service	- More verbose - Harder to develop, requires tools - Conceptually more difficult, more ”heavy-weight” than REST
REST	
Strengths (pros)	Weaknesses (cons)
+ Language and platform agnostic + Much simpler to develop than SOAP + Small learning curve, less reliance on tools + Concise, no need for additional messaging layer + Closer in design and philosophy to the Web + Suitable if scalability is a strong requirement + Protocol at the application level + Requirements that covers: scalability, asynchrony + Data as a service	- Assumes a point-to-point communication model - Not usable for distributed computing environment - Lack of standards support for security, etc. - Tied to the HTTP/HTTPS transport model - Not extensible

3. IMPLEMENTING PARALLEL AND DISTRIBUTED EAS. STATE OF THE ART

The inherent parallelism of EAs [36] has been widely studied in classical existing models (see, e.g., [20] for a survey) but mainly under the following approaches: master-slave, fine-grained and coarse-grained parallelization.

In the master-slave model (global parallelization or farming) [44,1,54], the algorithm runs on the master process and the individuals are sent for evaluation to the slave processes, in an approach usually called farming.

Fine-grained algorithms are often implemented on massively parallel computers. Usually, one individual is assigned to each processor, keeping tightly communication with the neighbors. In EA literature, they are also called cellular EAs [18].

In coarse-grained parallelization [82,70,21] (island model), the population is divided into small subpopulations that evolve in different processors. Each EA (island) process its population, and exchange some individuals with other islands with a certain rate and frequency [19,3,75]. Finally, some individuals are replaced at each island.

Recently, some implementations have been developed in which several islands use a shared pool through which all information is exchanged (i.e., individuals). Thus, Roy et al. [74] propose a shared memory multi-threaded architecture, in which several threads work independently on a single shared memory. Similar implementations propose using a database management system as storage [15,60], such as the public FluidDB platform [59].

Following the same scheme, but taking advantage of the current development of cloud architectures, some authors [5,13,6,9,10] propose using storage based cloud services, such as Dropbox.

Although these approaches work well on heterogeneous fully decentralized networks, the shared pool might represent a bottleneck.

As far as the implementation is concerned, nowadays there is a broad choice of parallel EA frameworks almost in every language, and adapted to every need. Most of them allow only the island model (DREAM [66,7], GALOPPS [50], Paladin-DEC [81]), while some of them implement master-slave architectures (PGAPack [32]). Only the most recent and updated support both models (ParadisEO [17], Distributed BEAGLE C++ [33,47], ECJ [85]). These libraries use compiled code, against the latest developments in which scripting languages (i.e., Perl and Python) are used, due to its flexibility and speed of implementation. Algorithm::Evolutionary (A::E) library [63,61] and EvoGrid [52] are included in this niche.

In most cases, these frameworks are initially created with a single problem in mind, and eventually expanded to a whole range of problems. Moreover, they use their own communications protocols (and even their data packing coding), and the user is forced to modify the source code or stop the execution to add new functionalities (like load balancing, dynamic control of operators, or an user interface).

Our proposal, as proposed in [68], focuses on using standard protocols and technologies in order to facilitate the communication and integration among them that can lead to parallel implementations with a high platform flexibility and interoperability.

```
use SOAP::Transport::HTTP;
my $daemon =
SOAP::Transport::HTTP::Daemon
    -> new (LocalPort => 80)
    -> dispatch_to('Demo');
$daemon->handle;

package Demo;
our $src="";
sub push {
    my ($class, $cad) = @_;
    $self->src = $cad;
    return "ok";
};
sub pop {
    my $tmp = $self->src;
    $self->src = " ";
    return $tmp;
};
```

```
use Time::HiRes qw( gettimeofday tv_interval);
use SOAP::Lite;
my $i=0;
my $tmp_it = [gettimeofday()];
for ($i=0; $i<100 ; $i++) {
    my $cad="01234567890 ... 01234567890";
    print SOAP::Lite
    -> uri('http://www.soaplite.com/Demo')
    -> proxy('http://vaio/')
    -> push($cad) -> result;
    print SOAP::Lite
    -> uri('http://www.soaplite.com/Demo')
    -> proxy('http://vaio/')
    -> pop() -> result;
};
print "TIME: ", tv_interval( $tmp_it );
```

Fig 1: SOAP programming example: server (left) and client (right). The value of the string $cad varies from 100 to 10000 characters in order to configure different loads.

4. EXPERIMENTAL SETUP AND RESULTS

In this paper we carry out three experiments to compare two parallel models implemented using SOAP and REST technologies in Perl language. The SOAP model was implemented using the SOAP::Lite (http://www.soaplite.com) module, while the REST implementation was carried out using the Perl Dancer (http://perldancer.org) module; both were chosen for their stability and maturity. In addition, servers developed using these modules are easy to implement and deployed using the computer infrastructure available to us in our department. As an example, Figures 1 and 2 show the Perl source code of the client-server SOAP and REST implementations developed for the first experiment (subsection 4.1).

The Experiment 1 (subsection 4.1) consisted in the implementation of a client-server model. In this case, the server process runs on a machine that attends client requests, involving different lengths of text strings. The experiment 2 (subsection 4.2) implements a master-slave based GA. In this case, a master process runs the GA, while different slave processes evaluate the fitness function. As experiment 3 (subsection 4.3), and based on the previous one, a real-coded EA to solve function optimization problems has been tested on three well known functions taken from the CEC2005 benchmark. The experiment 4 (subsection 4.4) implements a distributed EA to optimize MLPs: G-Prop method [22,23] is adapted as a distributed EA using SOAP and REST using a master-slave model.

Results shown in tables are obtained as the mean and standard deviation, and the experiment setup is such that the computational cost is the same for all proposed models and configurations, that is, different version of algorithms were run using the same parameter values.

```
use Dancer;
my $src = "";
get '/pop/' => sub {
    my $tmp = $src;
    $src = " ";
    return $tmp;
};
get '/push/:cad' => sub {
    my ($class, $cad) = @_;
    $src = $cad;
    return "ok";
};
Dancer->dance;
```

```
use Time::HiRes qw( gettimeofday tv_interval);
use LWP;
my $nav = new LWP::UserAgent;
$nav->agent("RESTzilla");
my $i=0;
my $tmp_it = [gettimeofday()];
for ($i=0; $i<100 ; $i++) {
    my $cad="01234567890 ... 01234567890";
    my $rpush = new HTTP::Request GET
    => 'http://127.0.0.1:3000/push/'."$cad";
    my $upush = $nav->request($rpush);
    my $rpop = new HTTP::Request GET
    => 'http://127.0.0.1:3000/pop/';
    my $upop = $nav->request($rpop);
};
print "TIME: ", tv_interval( $tmp_it );
```

Fig 2: REST programming example: server (left) and client (right). The string $cad value varies in size from 100 to 10000 characters in order to configure different loads.

Up to four computers have been used to run the algorithm and to obtain results both in sequential and parallel versions of the program. Experiments were conducted running the programs on a Windows-7 64bits and three Ubuntu/Linux (version 10, 32bits and version 12, 64bits) machines. Computer speeds range from 1.3 Ghz to 2 Ghz and are connected using the ethernet network of the university (with a high communication latency, i.e., an average *ping* of 7 ms). No experiments using homogeneous computer network have been done, because our aim is to demonstrate potential of distributed EA using web services on a distributed heterogeneous environment, taking advantage of flexibility.

4.1. Proof of Concept: Client-Server Efficiency Comparison

A classic client-server model is implemented in which clients can send and receive a text string. Different string lengths (100, 1000, 5000 and 10000 chars) have been configured in order to probe with different loads. In this way, we have tried to determine how the string length (the amount of data) affects the running time (due to communications).

The experiment consisted in sending 100 times an string of chars to the server (SOAP/REST). The experiment was repeated for 30 times for each case, measuring the time spent using *gettimeofday* function (in order to achieve a good precision). The best results have been highlighted in order to ease interpretation. In some cases, asterisks have been used to indicate significative differences regarding the other results.

As shown in Table 2, REST implementation is faster when sending small amount of data (100 and 1000 chars), while SOAP version takes similar time whenever the amount of data (differences between sending a 100 chars string and a 10000 chars string are smaller). The limit is about 5000 chars, when both implementations need similar time. Time taken to parse the XML messages in SOAP does not increment with the amount of data sent (as seen in Figure 3).

Results were verified using t-Student statistical tests and significant differences were found when the confidence level was 95% or 99%.

Table 2. Results obtained on the first experiment (client-server implementations). The best results are highlighted in order to ease interpretation (asterisks indicate significant differences regarding the other results).

	sending 100 chars	**sending 1000 chars**	**sending 5000 chars**	**sending 10000 chars**
SOAP	5.64 ± 0.17	5.83 ± 0.17	**5.61 ± 0.12 ***	**5.44 ± 0.08 ***
REST	**2.56 ± 0.10 ***	**3.45 ± 0.10 ***	5.82 ± 0.55	7.68 ± 0.59

ISSN 22773061

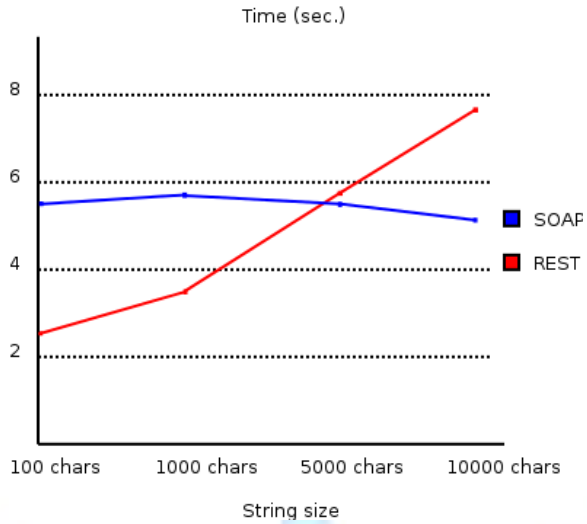

Fig 3: Time taken to complete the first experiment as the string size (amount of data) increases. About 5000 chars both implementations take a similar time.

4.2. Master-Slave based GA Implementation

In the Experiment 2, we have parallelized a GA following a master-slave model in order to solve a function optimization problem.

As stated before, an ideal client-server implementation of a distributed evolutionary algorithm could be a server process with several threads, in which each thread would include a population. However, as we cannot use a threaded version of the Perl modules, our implementation will focus on the fitness function evaluation.

Thus, the simplest way of task distribution along this model is to evaluate the individual fitness function on the slaves and to do the other steps on a master process (farming) as shown in Figure 4.

The evolutionary algorithm has been implemented using the A::E library [63,61], available under GPL license, in its version 0.76.2.

In this experiment, the fitness function is devoted to optimize the function given by equation 1, which is plotted in Figure 5. Our aim is to find the optimum (f (0, 0) = 1) with an accuracy of 10^{-6}. The experiment was repeated for 30 times for each configuration, measuring the time spent using *gettimeofday* function.

Para ver esta película, debe disponer de QuickTime™ y de un descompresor .

(1)

GA individuals are represented using bitstrings (data type A::E::Individual::BitString). As genetic operators, a bitflip mutation (A::E::Op::Mutation) and a two points crossover (A::E::Op::Crossover) are used.

Remainder GA parameter values are set as follows (default values found after an intensive experimentation are used, since we do not intend to find the optimal ones, but to prove feasibility of the implementation, and carry out a comparison):

- Population size = 50
- Generations = 20
- Individual length = 64 bits • Mutation rate = 20%
- Crossover rate = 80%
- Selection rate = 40%

The source code for all programs (servers, GA and evaluators), links to the A::E library and experiment data are available for download under GPL at http://geneura.ugr.es/ pedro/research/webservices

Table 3 shows the results obtained for this function optimization problem. The best results have been highlighted in order to ease interpretation, and asterisks have been used to indicate significative differences regarding the other results. As can be seen, the REST implementation is faster, due to the SOAP verbosity and time taken to decode the XML messages.

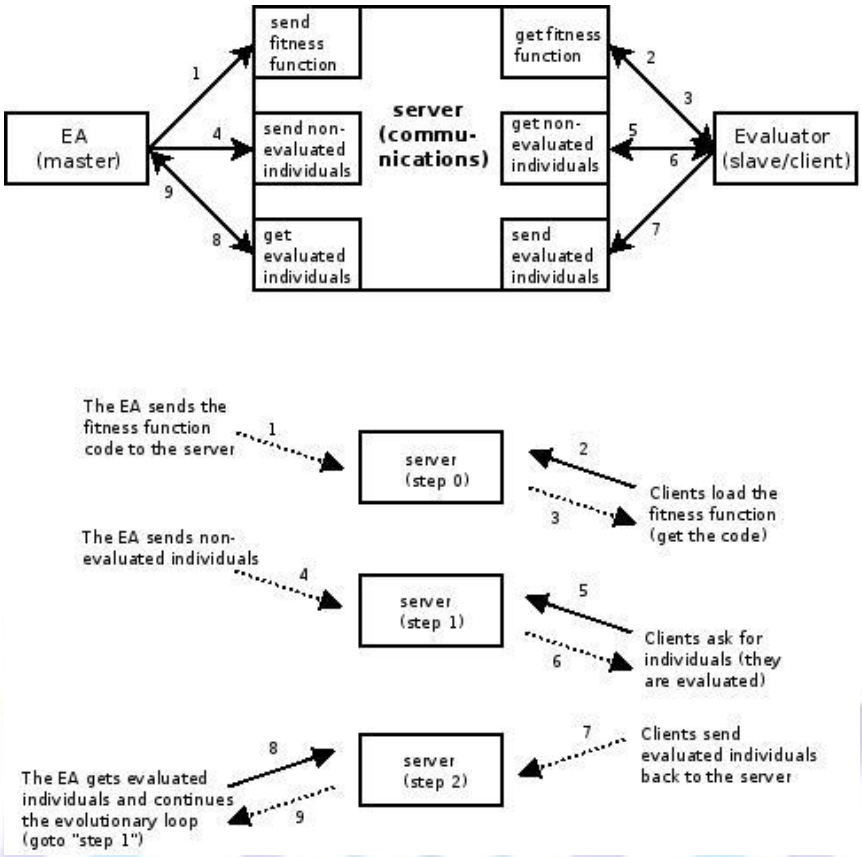

Fig 4: Scheme of the master-slave based implementation. The master process runs the EA/GA and the slave processes evaluate the fitness function. The dotted lines represent data transfers from or to the server, while solid lines refer to requests to the server.

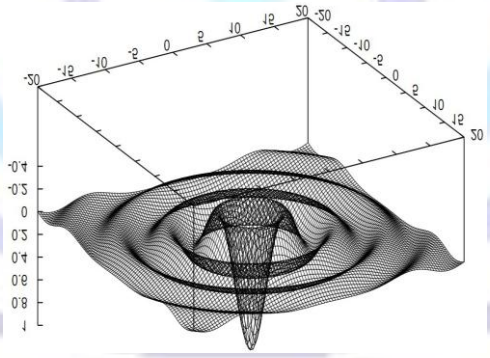

Fig 5: Fitness function representation (given by equation 1). The optimum of this function is f(0,0)=1

Table 3. Results obtained on the second experiment (master-slave implementations). The best results are highlighted in order to ease interpretation (asterisks indicate significant differences regarding the other results).

	10 generations **10 individuals**	**20 generations** **50 individuals**	**50 generations** **50 individuals**	**100 generations** **100 individuals**
SOAP accuracy	**0.997942 ± 0.000762***	**0.999867 ± 0.000101***	**1**	**1**
time (sec.)	3.79 ± 0.42	31.03 ± 1.89	133.08 ± 0.91	264.87 ± 0.39
REST accuracy	0.996092 ± 0.004081	0.999976 ± 0.000003	**1**	**1**
time (sec.)	**2.06 ± 0.08 ***	**15.05 ± 1.17 ***	**40.75 ± 2.81 ***	**100.84 ± 0.55 ***

ISSN 22773061

Both implementations obtain good results in terms of accuracy (both find the optimum with an accuracy of 10^{-6}) using even a small number of generations and population size. As far as the running time is concerned, REST implementation is faster in all configurations (see Table 3 and Figure 6). It might be due to the XML verboseness of SOAP communications (that increases the time taken to parse the messages). This result was expected taking into account the results obtained in the first experiment (subsection 4.1), as the message size in this experiment is small (64 chars). T-Student tests reported significant differences when the confidence level was 99%.

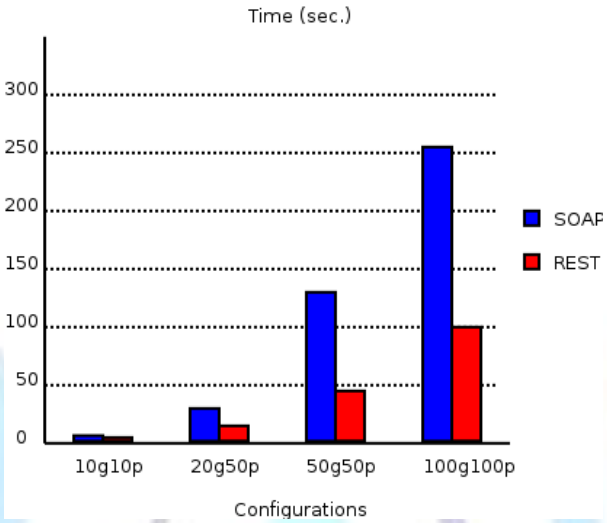

Fig 6: Comparing time (seconds) taken to complete the second experiment for each configuration (10 gen./10 indiv. ; 20 gen./50 indiv. ; 50 gen./50 indiv. ; 100 gen./100 indiv.).

4.3. Master-Slave based real coded EA Implementation

In this experiment, as in the previous one, we have parallelized an EA following a master-slave model in order to solve a function optimization problem using real coded individuals.

In this case, three well known function approximation problems, taken from the CEC2005 Competition [53], are used: F7 Shifted Rotated Griewangks Function, F8 Shifted Rotated Ackley's Function and F9 Shifted Rastrigin's Function. Detailed description of functions and conditions of testing are defined in [80], thus we provide only a brief information about the benchmark:

- The Griewangk function [51] is a continuous multimodal function with a high number of local optima. It is defined according to Equation 2 and its global optimum is located at point xi = 0, i = 1 : *Dimension* (n real numbers in the interval [0, 600]) [27].

Para ver esta película, debe disponer de QuickTime™ y de un descompresor .

(2)

- The Ackley function [2,14] (see Equation 3) is a multimodal non separable and regular function usually used as test function. The global optimum is located at point xi = 0, i = 1 : *Dimension* (n real numbers in the interval [−32, 32]).

Para ver esta película, debe disponer de QuickTime™ y de un descompresor .

(3)

- The Rastrigin function [83] (see Equation 4) is a multimodal real func- tion optimization problem, whose global optimum is located at point xi = 0, i = 1 : *Dimension* (n real numbers in the interval [−50, 50]) and whose minimum value is 0.

Para ver esta película, debe disponer de QuickTime™ y de un descompresor .

(4)

These problem functions have been addressed for 10D, 30D and 50D, and, in all cases, as the optimum is known, the fitness of an individual is calculated as the distance to the optimum for that function, and the goal is to obtain the smallest fitness for the optimized function.

EA individuals are represented using real number vectors (data type A::E::Individual::Vector). As genetic operators, a gaussian mutation (A::E::Op::GaussianMutation) and a two points crossover (A::E::Op::VectorCrossover) are used.

Remainder EA parameter values are set as follows (default values found after an intensive experimentation are used, since we do not intend to find the optimal ones, but to prove feasibility of the implementation, and carry out a comparison):

- Population size = 100
- Generations = 100
- Individual length = 10, 30 and 50 real numbers
- Mutation rate = 20%
- Crossover rate = 80%
- Selection rate = 40%

Table 4 shows obtained results for these function optimization problems. The mean value and standard deviation of the fitness function are reported, as well as the time taken to run the algorithms. The best results have been highlighted in order to ease interpretation, and asterisks have been used to indicate significative differences regarding the other results. As can be seen, the REST implementation is faster, due to the SOAP verbosity and time taken to decode the XML messages.

Table 4. Results obtained on the third experiment (master-slave implementations). The best results are highlighted in order to ease interpretation (asterisks indicate significant differences regarding the other results).

10D	F7		F8		F9	
	Fitness	Time	Fitness	Time	Fitness	Time
SOAP	**0.0002 ± 0.0001**	95.39 ± 84.49	**0.06 ± 0.04***	429.23 ± 73.39	**7.05 ± 4.42**	504.65 ± 74.21
REST	**0.0002 ± 0.0001**	**186.86 ± 45.34***	0.09 ± 0.05	**115.83 ± 56.82***	7.28 ± 5.30	**132.79 ± 64.13***
30D	F7		F8		F9	
	Fitness	Time	Fitness	Time	Fitness	Time
SOAP	**0.07 ± 0.06**	530.44 ± 69.24	**0.08 ± 0.09**	437.83 ± 54.82	**58.48 ± 31.04**	510.96 ± 62.24
REST	0.08 ± 0.05	**192.82 ± 35.68***	0.10 ± 0.12	**151.86 ± 38.06***	59.44 ± 25.22	**130.19 ± 59.16***
50D	F7		F8		F9	
	Fitness	Time	Fitness	Time	Fitness	Time
SOAP	**1.08 ± 0.63***	523.58 ± 39.08	**0.19 ± 0.16**	489.28 ± 85.38	**103.83 ± 89.35**	516.61 ± 75.87
REST	1.46 ± 0.91	**198.37 ± 63.43***	0.20 ± 0.17	**147.86 ± 53.04***	124.07 ± 95.55	**152.57 ± 68.09***

Both implementations obtain good results in terms of accuracy (both achieve re- sults close to the optimum) using even a small number of generations and population size. As far as the running time is concerned, REST implementation is faster due to the fact that no extra XML information is sent (that reduces the time taken to parse the messages) (see Table 4). This result was expected taking into account the results obtained in the first experiment (subsection 4.1), as the message size in this experiment is small, even for Dimension=50. T-Student tests reported significant differences when the confidence level was 99%.

As far as the convergence is concerned, Figure 7 shows the fitness evolution for both Griewangk, Rastrigin and Ackley (using as dimension 10, 30 and 50). In some cases premature convergence can be observed. This is due to the fact that no specific algorithm, selector nor genetic operators have been used to solve these problems (a canonical EA has been used).

The aim of this experiment is to use a costly and standard problem, focusing on comparing the performance of SOAP and REST implementations. In any case, good results have been obtained using both approaches, even comparable to those that can be found in the bibliography [79,39,72].

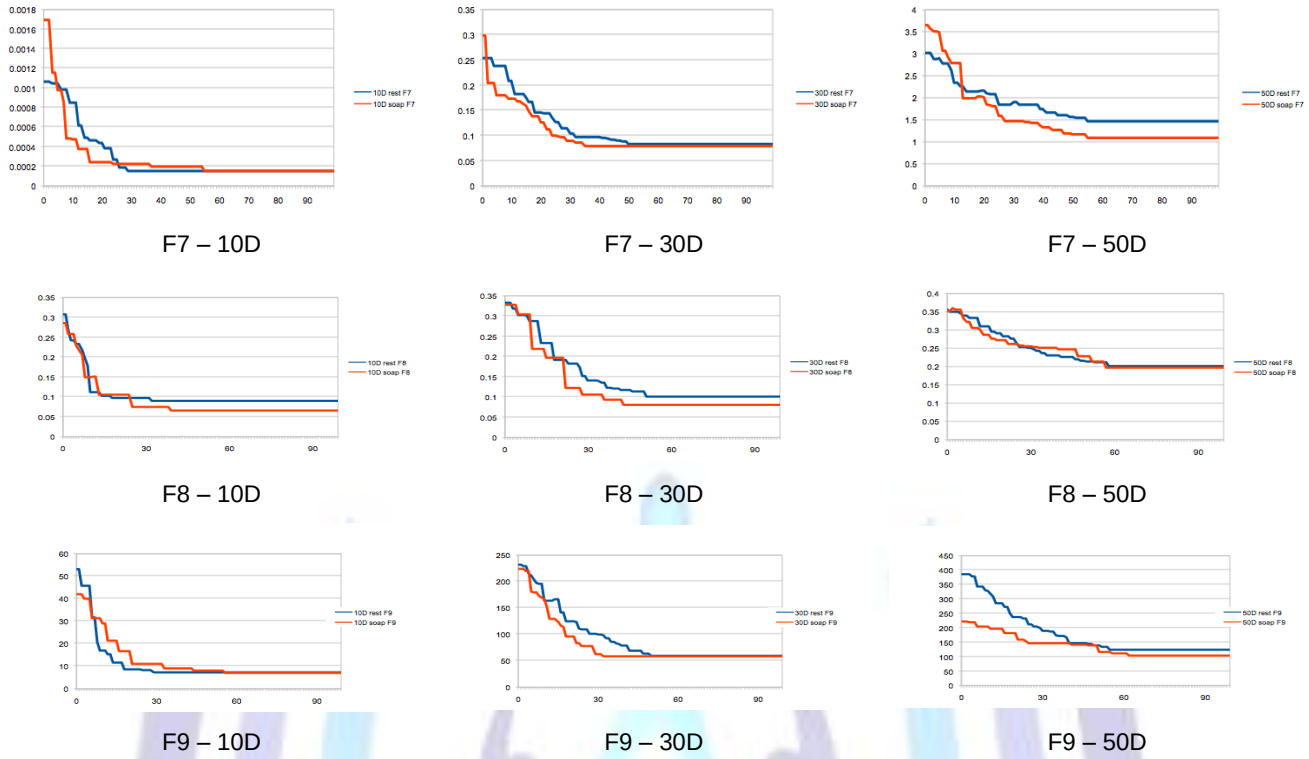

Fig 7: Evolution of fitness for the three functions and different configurations (10D, 30D and 50D). Each figure compares best fitness in each generation, for both SOAP (red) and REST (blue) implementations.

4.4. Master-Slave based EA implementation using Web-Services

In this experiment we adapt G-Prop as a distributed EA using SOAP and REST. Our implementation will focus on the most time consuming operation: the fitness function evaluation. G-Prop leverages the capabilities of two classes of algorithms: the ability of EA to find a solution close to the global optimum, and the ability of the back-propagation algorithm (BP) to tune a solution and reach the nearest local minimum by means of local search from the solution found by the EA. G-Prop method has been fully described and analysed out in previous papers (see [22,23]), thus we refer to these papers for further details. Instead of using a pre-established topology, the population is initialised with different hidden layer sizes. In most cases, evolved MLP should be coded into chromosomes to be handled by the genetic operators, however, G-Prop uses no binary codification, instead, the initial parameters of the network are evolved using specific variation operators such as mutation, multi-point crossover, addition and elimination of hidden units, and QP training applied as operator to the individuals of the population. The EA optimises the classification ability of the MLP, and at the same time it searches for the number of hidden units (architecture), the initial weight setting and the learning rate for that net.

The whole evolutionary algorithm is run on a master process and only the objective function is sent to the slaves for evaluation, following a *farming* model (as shown in Figure 4). The whole system can be sketched as follows:

1. The EA process sends the fitness function code to the server and creates the EA population.
2. Some clients connect the server and load the fitness function sent as Perl code from the server (Figure 2 shows an example of server and client processes implementations to upload and download the fitness function source code).
3. The EA process sends non-evaluated individuals to the server.
4. The clients ask for individuals to the server in order to evaluate them.
5. The clients evaluate individuals and send the result back to the server.
6. The EA process obtains evaluated individuals from the server and continues the evolutionary loop.
7. The EA terminates after a fixed number of generations (it sends a termination message throughout the server to the clients that remain ready to attend new workloads).

The server in these experiments is mainly used for scheduling and balancing the tasks among the different clients; the network itself is used for communication, but all the interchange of information among clients must be cleared by the central server. However, one of the objectives of the work presented in this paper has been to create an infrastructure that would get rid of the bottleneck represented by the central server in these experiments. As in the previous experiments, the EA has been implemented using the A::E library.

4.4.1. The experiment

The tests used to assess the accuracy (obtained error) of a method must be carefully selected, since some synthetic problems (also called “toy problems”) are not suitable for certain capacities of the BP algorithm, such as generalization

[37]. We agree with the view put forward by Prechelt [71], stating that real problems should be used in order to test an algorithm. In real life problems, the division between classes is not as clear as it is in synthetic problems. The dispersion of samples within a single class is also greater, due to noise [64]. In any case, the best way to test the algorithm ability as well as its limitations is to use it to resolve real world problems.

In this paper, the Glass pattern classification problem is used. This problem was put forward by Prechelt in his paper *"PROBEN1 – A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms"* [71].

This dataset is based on the glass problem dataset from the UCI library of machine learning databases. This task was prompted by the needs of forensic scientists involved in criminal investigation. The results of a chemical analysis of glass splinters (content of 8 different elements in percentage terms) together with a refractive index, are used in the classification of the sample as either float-processed or non-float processed building windows, vehicle windows, containers, tableware or head lamps. It contains 214 entries. Each sample has 9 attributes plus the class attribute (type of glass): refractive index, sodium, magnesium, aluminium, silicon, potassium, calcium, barium, and iron. This dataset is very difficult to classify due to two important features. First, the number of available patterns is low (214) for six different classes. Second, the number of patterns in each class is very unbalanced, ranging from 76 (building windows non-float processed) to 9 (tableware).

Dataset was divided into three disjoint parts: one for training, one for validating, and one for testing, as proposed in [71]. In order to determine the fitness of an individual, the MLP was trained by the training set and its fitness was established from the classification error with the validating set. After the EA has finished, i.e., when it has reached the limit of generations, we obtain the generalization ability by using the testing set (previously unseen patterns). This generalization value is shown in tables.

On the other hand, it is very important to know which parameter values involved in the design of an EA have the greatest influence on its behaviour and performance. Adjusting the main design parameters has been usually solved either by hand [31,55,29] or using conventions, ad-hoc choices, intensive experimentation with different values [35,77,34,76,78], and even random initialization values [49]. As Eiben states, the straightforward approach is to generate and test [57,34,76]. Nevertheless, when making a detailed statistical analysis of the influence of each parameter, the designer should pay greater attention to the parameter providing the values that are statistically most significant [24].

Table 5. Parameters set using statistical methods. The number of generations and the population size needed for greater diversity should, of course, be higher or lower depending on the difficulty of the problem.

Parameter	Value
number of generations	500
population size	500
selection rate	20%
initial weights range	[−0.05, 0.05]
mutation operator priority	2.0
crossover operator priority	0.5
addition operator priority	1.0
elimination operator priority	0.5
training operator priority	0.5
mutation probability	0.4
weight mutation range	[−0.001, 0.001]
learning constant mutation range	[−0.010, 0.010]

Authors carried out a statistical study [26] in order to determine the most important parameters (regarding their influence on the results), and to establish the most suitable values for such parameters (thus obtaining an optimal operation). In this study, the ANOVA (ANalysis Of the VAriance) [42,43] statistical method was used. This statistical tool, based on the analysis of the mean variance, is widely used. The ANOVA method was used to determine whether a change in the responses is due to a change in a factor or due to a random effect. Besides the ANOVA method, the statistical analysis tool ANOM (ANalysis Of Mean) was used. This technique uses the average values for each parameter level (value) and the main effect (on the responses) plots, to decide the most suitable values for each parameter. As a result, running parameter values shown in Table 5 were obtained.

The number of generations and the population size needed for greater diversity should, of course, be higher or lower depending on the difficulty of the problem.

Time taken to run the EA was measured using the *gettimeofday* function, in order to achieve a good precision, from which mean and standard deviations (for 30 runs) shown in Table 6 were obtained. Sequential version of the program was run in the faster machine; and in parallel runs, the EA (master process) was run on the faster machine while the evaluators were run on slower machines.

4.4.2. Obtained results

The aim of this experiment is to use a costly problem that justifies using a farming model, but always with an emphasis on comparing the performance of SOAP and REST implementations. Results obtained can be shown in Table 6. The best results have been highlighted in order to ease interpretation. In some cases, asterisks have been used to indicate significative differences regarding the other results.

Table 6. Results (error % and time) obtained using both the sequential and the parallel versions (comparing using the same number of evaluations). Up to 4 evaluators-slaves are used in the farming model. The best results are highlighted in order to ease interpretation (asterisks indicate significant differences regarding the other results).

Model	Error (%)	Time (seconds)
Sequential	33 ± 2	**1215 ± 104 ***
Master-slave (using SOAP)		
1 eval.	32.8 ± 1.3	1343 ± 116
2 eval.	**32.2 ± 1.5**	**717 ± 81***
3 eval.	31.8 ± 1.3	**508 ± 91***
4 eval.	**31.0 ± 1.6 ***	**404 ± 89***
Master-slave (using REST)		
1 eval.	**32.2 ± 1.6 ***	1517 ± 109
2 eval.	32.4 ± 1.5	804 ± 89
3 eval.	**31.4 ± 2.1**	566 ± 97
4 eval.	31.6 ± 1.8	447 ± 82

Table 6 and Figure 8 show the generalization performance on previously unseen patterns, as well as the running time. As can be seen, classification errors show a comparable algorithmic result, as there are small differences on obtained error, although running time reduces when dividing the problem between several computers that work in parallel.

Results were verified using t-Student statistical tests. In the case of classification error, significant differences were found when the confidence level was 80%. In some cases, the means were so similar that no significant differences were found (using 2 and 3 evaluators). In all cases, the resultant times presented significant differences when the confidence level was 95% (using 3 and 4 evaluators) and even as high as 99% for 1 and 2 evaluators. Note that the sequential model achieves a lower time, due to the fact that no communication overload is introduced in this case.

As can be seen in Figure 9, SOAP implementation is slightly faster than the REST one, as the amount of data sent in each communication is very high (more than 6000 chars on average). This result is in agreement with the first experiment, where for large amounts of data (big messages), REST communications take longer than SOAP communications.

Figure 10 shows the speedup obtained in these runs. We can see that classification ability and running time can be improved dividing the problem between several processes that work in parallel. Speedup does not equals the number of computers used; however, running time is improved using several computers. Thus, as adding new evaluators (heterogeneous computers running a Perl process) is an easy and costless task, we could take advantage of this system structure and flexibility to solve costly optimization problems. In the case of SOAP implementation, speedup is slightly better than in the case of REST, due to the fact that SOAP communications are faster for big messages.

Fig 8: Plot of the classification error (%). As can be seen, the results in error are very similar, what means that both techniques are suitable for developing parallel systems.

ISSN 22773061

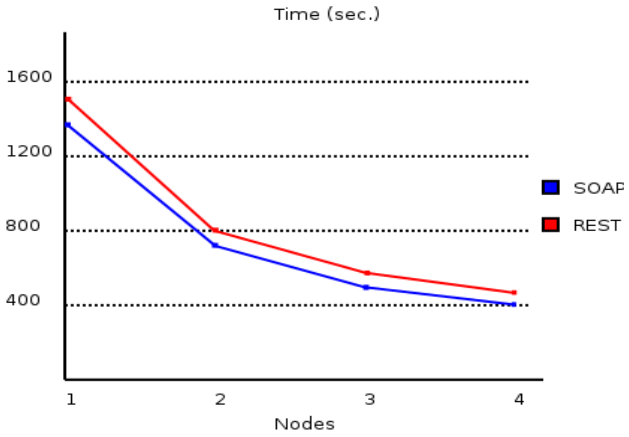

Fig 9: Plot of the running time (seconds) for both the SOAP and REST implementations.

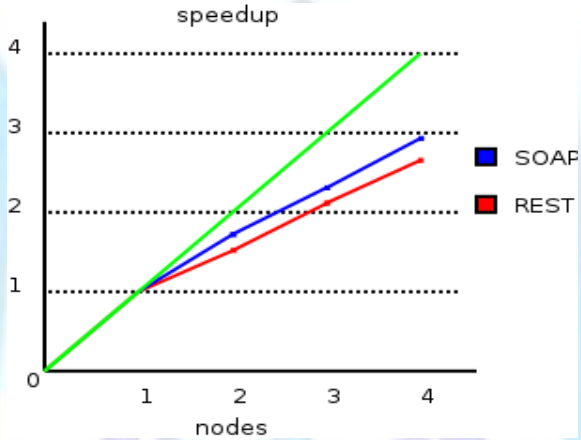

Fig 10: Plot of the speedup obtained using SOAP and REST implementations, and f(x) = x function (solid line). Although speedup is not lineal, it can be seen that running time is improved using several computers as clients dedicated to evaluate the individual fitness functions.

5. CONCLUSIONS

This paper presents a new parallel-distributed computation implementation using SOAP and REST web services that shows the useful these technologies can be in the field of evolutionary computation.

As reported in the experiments provided, both techniques are suitable for developing parallel systems. However, depending on the amount of data sent, one might be heavier than the other.

As seen, to implement and use communications using SOAP and REST it is not necessary running virtual machines (as in Java programming), nor daemons, just only to install several libraries available for almost any programming language. Moreover, an arbitrary number of computers (clients-evaluators) can be added to the system, making it faster.

In these experiments, we have demonstrated that both can be used as communication protocol for distributed evolutionary computation, obtaining a good speedup. Results obtained are comparable, and only for big messages, REST communications take longer than SOAP communications. Differences in time between the proposed approaches are significant after the application of t-Student statistical tests. They provide a common interface that can be called from almost any programming language and under any platform, taking advantage of their flexibility, fault tolerance and resilience to infrastructure breakdowns. Thus, programs can be written in any language and can share data without the need of worrying about the message formats or communication protocols.

Using REST and SOAP could increase the network overload, mainly because the headers size. For example, minimum header size in SOAP is 314 bytes (HTTP headers and SOAP envelope). However, this amount of data does not overload the network; moreover, in the case of sending a large number of data, this header size represents a small percentage of the total. On another hand, using either SOAP or REST provides several advantages: Protocols such as SOAP provide access to services. Besides, using the standard HTTP ports also skips network and firewall configuration (no new ports need to be open). Thus, SOAP and REST should be used in case the data to transmit is large enough (such as the experiments presented), where the header size is a small percentage of the data sent. In any case, while running these experiments, no overload on the network was noticed (no delays in communications). Using other distributed systems, such as Jini [12], the network traffic is so high that when a high number of computers are used, communication becomes difficult.

From these implementations, several paths for improvement are devised: changing the models so that more computation is moved to the clients, leaving the server as just a hub for information interchange among clients; that information interchange will have to be reduced to the minimum. That will make this model closer to the island model, with just the migration policies regulated by the server. That way, the server bottleneck is almost eliminated.

Finally, it is necessary to use flexible tools that allow EA researchers to take advantage of all available computation nodes, and make possible the self-adaptation of EAs in every execution environment and for each problem type. In this sense, the EA researchers should start to consider a migration from their actual software to be accessed as services. Thus, as future research, it could be of interest adding support for SOAP and REST to existing distributed evolutionary algorithm libraries, such as JEO [8], EO [56], and libraries in other languages, in order to allow the implementation of multi-language evolutionary algorithms. Another possibility is to test P2P architectures, where each computer communicates only with one or two computers in the network. It would be very interesting to parallelize the proposed method using random topologies, in such a way that a ”servent” (server/client) can enter or leave the network at any moment.

ACKNOWLEDGEMENTS

This work has been supported in part by EvOrq (P08-TIC-3903), CANUBE (CEI2013-P-14) and ANYSELF (TIN2011-28627-C04-02) projects. The authors would like to thank the FEDER of European Union for financial support via project "Sistema de Información y Predicción de bajo coste y autónomo para conocer el Estado de las Carreteras en tiempo real mediante dispositivos distribuidos" (SIPEsCa) of the "Programa Operativo FEDER de Andalucía 2007-2013". We also thank all Agency of Public Works of Andalusia Regional Government staff and researchers for their dedication and professionalism.

REFERENCES

[1] J. Abramson and A. Abela. 1992. Parallel genetic algorithm for solving the school timetabling problem. In Proceedings of the Fifteenth Australian Computer Science Conference (ACSC-15), vol. 14, p.1-11.

[2] D. H. Ackley. 1987. A connectionist machine for genetic hillclimbing. Kluwer, Boston.

[3] E.Alba and J.M.Troya. 2000. Influence of the Migration Policy in Parallel Distributed GAs with Structured and Panmictic Populations. Applied Intelligence Volume 12, Number 3, 163-181, DOI: 10.1023/A:1008358805991.

[4] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. 2010. Web Services: Concepts, Architectures and Applications. Springer. ISBN 978-3642078880.

[5] A. Andrzejak, D. Kondo, and D. Anderson. 2010. Exploiting non-dedicated resources for cloud computing. 12th IEEE/IFIP Network Operations and Management Symposium (NOMS 2010), Osaka, Japan. pp. 341-348.

[6] M.G. Arenas, P.A. Castillo, G. Romero, and J.J. Merelo. 2011. Cloud-based Evolutionary Parallel Computation using low cost storage services. IEEE First International Symposium on Network Cloud Computing and Applications. isbn: 978-0-7695-4550- 9/11. pp.56-61. 2011. DOI 10.1109/NCCA.2011.16.

[7] M.G. Arenas, Pierre Collet, A.E. Eiben, Mark Jelasity, J. J. Merelo, Ben Paechter, Mike Preuß, and Marc Schoenauer. 2002. A framework for distributed evolutionary algorithms. Number 2439 in Lecture Notes in Computer Science, LNCS, pages 665–675. Springer-Verlag, September.

[8] M.G. Arenas, L. Foucart, J.J. Merelo-Guervós, and P. A. Castillo. 2001. JEO: a framework for Evolving Objects in Java. In Actas Jornadas de Paralelismo, pages 185–191. UPV, Universidad Politécnica de Valencia.

[9] M.G. Arenas, J.J. Merelo, A.M. Mora, P.A. Castillo, G. Romero, and J.L.J. Laredo. 2011. Assessing speed-ups in commodity cloud storage services for distributed evolutionary algorithms. 2011 IEEE Congress on Evolutionary Computation. isbn 978-1-4244-7833-0/11 pp.304-311.

[10] M.G. Arenas, J.J. Merelo, A.M. Mora, P.A. Castillo, G. Romero, and J.L.J. Laredo. 2011. Using free cloud storage services for distributed evolutionary algorithms. 13th An- nual Genetic and Evolutionary Computation Conference, GECCO 2011. Proceedings, Dublin, Ireland, July 12-16, 2011. Natalio Krasnogor and Pier Luca Lanzi, Editors. ACM. pages 1603-1610. isbn 978-1-4503-0557-0.

[11] A. Arsanjani, S.Ghosh, A.Allam, T.Abdollah, S.Ganapathy and K.Holley. 2008. SOMA: A method for developing service-oriented solutions. IBM Systems Journal, 47(3):377-396.

[12] J. Atienza, M. García, J. González, and J.J. Merelo-Guervós. 2000. Jenetic: a distributed, fine-grained, asynchronous evolutionary algorithm using Jini. In P. P. Wang, editor, Proc. JCIS 2000 (Joint Conference on Information Sciences), volume I, pages 1087-1089. ISBN: 0-9643456-9-2.

[13] M. Atkinson, D. D. Roure, J. van Hemert, and D. Michaelides. 2010. Shaping ramps for data-intensive research. UK e-Science All Hands Meeting 2010.

[14] T. Bäck. 1996. Evolutionary algorithms in theory and practice. Oxford University Press.

[15] A. Bollini and M. Piastra. 1999. Distributed and persistent evolutionary algorithms: a design pattern. In Genetic Programming, Proceedings EuroGP99. Lecture Notes in Computer Science, no. 1598. Springer, pp. 173-183.

[16] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H.F. Nielsen, S. Thatte, and D. Winer. 2011. Simple Object Access Protocol (SOAP) 1.1, W3C Note 08 May 2000. Available from http://www.w3.org/TR/SOAP.

[17] S. Cahon, N. Melab, and E. Talbix. 2004. ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. Journal of Heuristics, pp. 357-380.

[18] G. Cantor and J. Gómez. 2010. Maintaining genetic diversity in fine-grained parallel genetic algorithms by combining cellular automata, Cambrian explosions and massive extinctions. 2010 IEEE Congress on Evolutionary Computation (CEC), pp.1-8, ISBN 978-1-4244-6909-3.

[19] E. Cantú-Paz. 1999. Topologies, migration rates, and multi-population parallel genetic algorithms. In: Banzhaf W, Daida J, Eiben AE, Garzon MH, Honavar V, Jakiela M, Smith RE (eds) Proceedings of the genetic and evolutionary computation conference, vol 1, Orlando, Florida, USA, 1317 July 1999. Morgan Kaufmann, London, pp 9198.

[20] E. Cantú-Paz. 2000. Efficient and accurate parallel genetic algorithms. Kluwer Academic, Norwell.

[21] E. Cantú-Paz and D. E. Goldberg. 1997. Modeling idealized bounding cases of parallel genetic algorithms. In Koza J., Deb K., Dorigo M., Fogel D., Garzón M., Iba H., Riolo R. Eds. Genetic Programming 1997: Proceedings of the Second Annual Conference, Morgan Kaufmann (San Francisco. CA).

[22] P. A. Castillo, J. Carpio, J. J. Merelo, V. Rivas, G. Romero, and A. Prieto. 2000. Evolving multilayer perceptrons. Neural Processing Letters, 12:115–127.

[23] P. A. Castillo, Juan-Julin Merelo-Guervós, A. Prieto, V. Rivas, and G. Romero. 2000. G-Prop: Global optimization of multilayer perceptrons using GAs. Neurocomputing, 35:149–163.

[24] P.A. Castillo, M.G. Arenas, N. Rico, A.M. Mora, P. García, J.L.J. Laredo, and J.J. Merelo. 2012. Determining the significance and relative importance of parameters of a simulated quenching algorithm using statistical tools. Applied Intelligence Volume 37, Number 2, 239-254, DOI: 10.1007/s10489-011-0324-x.

[25] P.A. Castillo, P. García-Sánchez, M.G. Arenas, J.L. Bernier, and J.J. Merelo. 2012. Distributed evolutionary computation using soap and rest web services. In Joanna Kolodziej, Samee Ullah Khan, and Tadeusz Burczynski, editors, Advances in Intelligent Modelling and Simulation, volume 422 of Studies in Computational Intelligence, pages 89–111. Springer Berlin Heidelberg.

[26] P.A. Castillo, J.J. Merelo, G. Romero, A. Prieto, and I. Rojas. 2002. Statistical Analysis of the Parameters of a Neuro-Genetic Algorithm. in IEEE Transactions on Neural Networks, vol.13, no.6, pp.1374-1394, ISSN:1045-9227, IEEE Press.

[27] H. Cho, F. Olivera, and S.D. Guikema. 2008. A derivation of the number of minima of the griewank function. Applied Mathematics and Computation, 204(2):694–701.

[28] Robert Daigneau. 2011. Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services . Addison-Wesley Professional. 1 Ed. ISBN 978-0321544209.

[29] Swagatam Das, Archana Chowdhury, and Ajith Abraham. 2009. A bacterial evolutionary algorithm for automatic data clustering. In Proceedings of the Eleventh conference on Congress on Evolutionary Computation, CEC'09, pages 2403–2410, Piscataway, NJ, USA. IEEE Press.

[30] S.B. Davidson and J. Freire. 2008. Provenance and scientific workflows: challenges and opportunities. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 1345–1350. ACM.

[31] L. Davis. 1991. Handbook of genetic algorithms. Van Nostrand Reinhold, NY.

[32] J. Digalakis and K. Margaritis. 1999. An Experimental Study of Genetic Algorithms using PGAPack. In Proceedings of 7th Hellenic Conference on Informatics, volume 1, pages v34–v40, Ioannina. University of Ioannina, E.P.Y.

[33] Marc Dubreuil, Christian Gagne, and Marc Parizeau. 2006. Analysis of a master-slave ar- chitecture for distributed evolutionary computations. IEEE Transactions on Systems Man and Cybernetics Part B, 36(1):229–235.

[34] A. E. Eiben. 2009. Principled Approaches to tuning EA parameters. Tutorials - IEEE Congress on Evolutionary Computation (CEC2009). Available at http://www.few.vu.nl/˜gusz/papers/eiben-cec-2009-tutorial-corrected.pdf

[35] A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith. 2007. Parameter control in evolutionary algorithms. Parameter Setting in Evolutionary Algorithms, Studies in Computational Intelligence. Volume 54/2007, pp.19-46. ISBN 978-3-540-69431-1, ISSN 1860-949X. Springer Berlin / Heidelberg.

[36] A.E. Eiben and J.E. Smith. 2003. Introduction to evolutionary computing. Springer, New York.

[37] S.E. Fahlman. 1988. An empirical study of learning speed in back-propagation networks. Technical report, Carnegie Mellon University.

[38] S.E. Fahlman. 1988. Faster-Learning Variations on Back-Propagation: An Empirical Study. Proceedings of the 1988 Connectionist Models Summer School, Morgan Kaufmann.

[39] S.K. Fan and J.M. Chang. 2010. Dynamic multi-swarm particle swarm optimizer using parallel PC cluster system for global optimization of large-scale multimodal functions. Engineering Optimization, Vol. 42, No. 5, pp. 431-451.

[40] R.T. Fielding. 2000. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine.

[41] R.T. Fielding and R.N. Taylor. 2002. Principled Design of the Modern Web Architecture. ACM Transactions on Internet Technology (TOIT) (New York: Association for Computing Machinery) 2 (2): 115-150.

[42] R.A. Fisher. 1925. Theory of Statistical Estimation. Proceedings of the Cambridge Philosophical Society, 22, pp.700-725.

[43] R.A. Fisher. 1936. The Comparison of Samples with Possibly Unequal Variances. Annals of Eugenics, 9, pp.174-180.

[44] T. Fogarty and R. Huang. 1991. Implementing the genetic algorithm on transputer based parallel processing systems. Parallel Problem Solving From Nature,p.145-149.

[45] I. Foster, K. Czajkowski, DE Ferguson, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke. 2005. Modeling and managing state in distributed systems: The role of ogsi and wsrf. Proceedings of the IEEE, 93(3):604–612.

[46] C. Gagné and M. Parizeau. 2006. Genericity in evolutionary computation software tools: Principles and case-study. International Journal on Artificial Intelligence Tools 15(2):173.

[47] Christian Gagn, Marc Parizeau, and Marc Dubreuil. 2003. Distributed beagle: An environment for parallel and distributed evolutionary computations. In Proc. of the 17th Annual International Symposium on High Performance Computing Systems and Ap- plications (HPCS) 2003, pages 201–208.

[48] P. García-Sánchez, J. González, P.A. Castillo, J.J. Merelo, A.M. Mora, J.L.J. Laredo, and M.G. Arenas. 2010. A distributed service oriented framework for metaheuristics using a public standard. In Nature Inspired Cooperative Strategies for Optimization, NICSO 2010, May 12-14, 2010, Granada, Spain. Studies in Computational Intelligence, volume 284, pages 211–222.

[49] Y. Gong and A. Fukunaga. 2011. Distributed Island-Model Genetic Algorithms Using Heterogeneous Parameter Settings. Proceedings of the 2011 IEEE Congress on Evolution- ary Computation (CEC2011), pp. 820-827. ISBN 978-1-4244-7834-7. New Orleans, LA, USA.

[50] E. D. Goodman. 1996. An introduction to GALOPPS - The Genetic Algorithm Optimized for Portability and Parallelism System. Release 3.2, Technical Report 96-07-01, In- telligent Systems Laboratory and Case Center for Computer-Aided Engineering and Manufacturing, Michigan State University, East Lansing, Mich.

[51] A.O. Griewank. 1981. Generalized descent for global optimization. Journal of Optimization Theory and Applications, 34(1):11–39.

[52] Olivier Grisel. 2012. EvoGrid. Distributed Evolutionary Computation framework. http://pypi.python.org/pypi/evogrid

[53] N. Hansen. 2006. Compilation of results on the 2005 CEC benchmark function set. http://bit.ly/11Rt5hS

[54] R. Hauser and R . Männer. 1994. Implementation of standard genetic algorithm on mimd machines. In Davidor Y., Schwefel H. P., Manner R., Eds., Parallel Problem Solving from Nature, PPSN III, p. 504-513, Springer-Verlag (Berlin).

[55] I. Jagielska, C. Matthews, and T. Whitfort. 1999. An investigation into the application of neural networks, fuzzy logic, genetic algorithms, and rough sets to automated knowledge acquisition problems. Neurocomputing, Vol. 24, no.1-3, pp.37-54.

[56] M. Keijzer, J.J. Merelo, G. Romero, and M. Schoenauer. 2002. Evolving Objects: a general purpose evolutionary computation library. In Pierre Collet, Cyril Fonlupt, Jin-Kao Hao, Evelyne Lutton, and Marc Schoenauer, editors, Artificial Evolution, 5th International Conference, Evolution Artificielle, EA 2001, Le Creusot, France, October 29-31, 2001, Selected Papers, volume 2310 of Lecture Notes in Computer Science, pages 231–244. Springer.

[57] D. Kim and C. Kim. 1997. Forecasting time series with genetic fuzzy predictor ensemble. IEEE Transactions on Fuzzy Systems, vol.5,no.4,pp.523-535.

[58] Bertram Ludascher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. 2006. Scientific workflow management and the kepler system. Concurrency and Computation: Practice and Experience, 18(10):1039–1065.

[59] J. Merelo. 2010. Fluid evolutionary algorithms. 2010 IEEE Congress on Evolutionary Computation (CEC2010). IEEE Press, pp. 18.

[60] J. Merelo, P.A. Castillo, J. Laredo, A. Mora, and A. Prieto. 2008. Asynchronous distributed genetic algorithms with Javascript and JSON. In WCCI 2008 Proceedings. IEEE Press, pp. 1372-1379.

[61] J.J. Merelo, P.A. Castillo, and E. Alba. 2010. Algorithm::Evolutionary, a flexible Perl module for evolutionary computation. Soft Computing. A Fusion of Foundations, Methodologies and Applications. Springer Berlin / Heidelberg. issn 1432-7643. Vol.14, issue 10, pp. 1091-1109.

[62] J.J. Merelo, G. Romero, M.G. Arenas, P.A. Castillo, A.M. Mora, and J.L.J. Laredo. 2011. Implementation matters: Programming best practices for evolutionary algorithms. In Joan Cabestany, Ignacio Rojas, and Gonzalo Joya, editors,

Advances in Computational Intelligence, volume 6692 of Lecture Notes in Computer Science, pages 333–340. Springer Berlin Heidelberg.

[63] J.J. Merelo. 2010. A perl primer for evolutionary algorithm practitioners. SIGEVOlution, 4(4):12–19.

[64] J.J. Merelo, A. Prieto, and F. Morán. 2001. Optimization of classifiers using genetic algorithms, chapter 4, pages 91–108. MIT press. ISBN: 0262162016.

[65] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M.R. Pocock, A. Wipat, and P. Li. 2004. Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics, 20(17):3045–3054.

[66] B. Paechter., T. Bck, M. Schoenauer, M. Sebag, A. E. Eiben, J. J. Merelo, and T. C. Fogarty. 2000. A distributed resource evolutionary algorithm machine (DREAM). In Evolutionary Computation, 2000. Proceedings of the 2000 Congress on, volume 2, pages 951–958. IEEE, IEEE Press.

[67] Mike Papazoglou and Willem-Jan van den Heuvel. 2007. Service oriented architectures: approaches, technologies and research issues. The VLDB Journal, 16:389–415. 10.1007/s00778-007-0044-3.

[68] J. Parejo, A. Ruiz-Cortes, S. Lozano, and P. Fernández. 2012. Metaheuristic optimization frameworks: a survey and benchmarking. Soft Computing - A Fusion of Foundations, Methodologies and Applications, 16:527-561 10.1007/s00500-011-0754-8.

[69] S. Perera and D. Gannon. 2006. Enabling web service extensions for scientific workflows. In Workflows in Support of Large-Scale Science, 2006. WORKS'06. Workshop on, pages 1–10. IEEE.

[70] C. B. Pettey, M. R. Leuze, and J. J. Grefenstette. 1987. A parallel genetic algorithm. In J. J. Grefenstette (Ed.), Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, pp. 155-162.

[71] L. Prechelt. 1994. PROBEN1 - A set of benchmarks and benchmarking rules for neural network training algorithms. Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany.

[72] A.K. Qin, V.L. Huang, and P.N. Suganthan. 2009. Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization. IEEE Transactions on Evolutionary Computation, Vol. 13, No. 2.

[73] Erik T. Ray. 2001. Learning XML: creating self-describing data. O´Reilly.

[74] G. Roy, H. Lee, J. Welch, Y. Zhao, V. Pandey, and D. Thurston. 2009. A distributed pool architecture for genetic algorithms. IEEE Congress on Evolutionary Computation, 2009. CEC 09. pp.1177-1184.

[75] C. Salto and E. Alba. 2012. Designing heterogeneous distributed GAs by efficiently self-adapting the migration period. Applied Intelligence Volume 36, Number 4, 800-808, DOI: 10.1007/s10489-011-0297-9.

[76] S. K. Smit and A. E. Eiben. 2009. Comparing parameter tuning methods for evolutionary algorithms. In CEC'09: Proc. of the Eleventh conference on Congress on Evolutionary Computation, pages 399–406, Piscataway, NJ, USA. P. Haddow et al. (Eds.), IEEE Press.

[77] S. K. Smit and A. E. Eiben. 2009. Using Entropy for Parameter Analysis of Evolutionary Algorithms. in Bartz Beielstein et al. (eds.), Empirical Methods for the Analysis of Optimization Algorithms, Natural Computing Series, Springer.

[78] S.K. Smit and A.E. Eiben. 2010. Beating the world champion Evolutionary Algorithm via REVAC Tuning. In Proc. of the WCCI2010 IEEE World Congress on Computational Intelligence, pp. 1295-1302. issn 978-1-4244-8126-2. IEEE Press. Barcelona, Spain.

[79] C. Song, H. Zhao, W. Cai, H. Zhang, and M. Zhao. 2007. The limited mutation particle swarm optimizer. K. Li et als (eds). LSMS 2007, LNCS Vol 4688, pp. 258-266, Springer-Verlag Berlin Heidelberg.

[80] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.P. Chen, A. Auger, and S. Tiwari. 2005. Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. Technical Report, Nanyang Technological University, Singapore, May 2005 AND KanGAL Report 2005005, IIT Kanpur, India.

[81] K.C. Tan, A. Tay, and J. Cai. 2003. Design and implementation of a distributed evolutionary computing software. IEEE Transactions on Systems, Man, and Cybernetics, Part C, vol. 33, no. 3, pp. 325-338.

[82] R. Tanese. 1987. Parallel genetic algorithms for a hypercube. In J. J. Grefenstette (Ed.), Proceedings of the second International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, pp. 177-184.

[83] A. To¨rn and A . Zilinskas. 1989. Global Optimization. Vol. 350 of Springer Lecture Notes in Computer Science (Springer, Berlin), pp. 1–24.

[84] S. Vinoski. 2008. Serendipitous reuse. IEEE Internet Computing, 12(1):84-87.

[85] David White. 2012. Software review: the ECJ toolkit. Genetic Programming and Evolvable Machines, 13:65–67.

ISSN 22773061

Author' biography with Photo

Pedro A. Castillo-Valdivieso received the B.Sc. degree in Computer Science and the Ph.D. degree, both from the University of Granada, Spain, in 1997 and 2000. He was a Visiting Researcher at the Napier University, Edinburg, U.K., in July 1998 and at the Santa Fe Institute, Santa Fe, NM, in September 2000. He has been leader of several local research projects, and directed four PhDs. His main interests include the optimization of artificial neural networks using evolutionary algorithms. He was a Teaching Assistant at Computer Science Department of the University of Jaén, Spain. He is currently Associate Professor at the Computer Architecture and Technology Department, University of Granada.

Pablo García Sánchez is a PhD Student in University of Granada (Spain). He received his Deegre (2007) and MsC (2008) in Computer Science and Engineering at the same University. He has participated in several national research projects related with e-Health, web-services and optimization, whose results have been published in conferences such as PPSN, EVO* and IWANN. His interests include Service Oriented Computing, evolutionary computation and distributed algorithms.

Maribel García Arenas, PhD in Computer Science and lecturer at the University of Granada; she's mainly interested in parallel and distributed computation and has been leader of several local research and innovation projects. During the last years she has several papers published in important conferences like PPSN, GECCO and CEC and two indexed papers have been accepted for publishing in two "Journal Citation Reports" journals. She has demonstrated experience with other subjects like Neural Networks or Evolutionary Computation.

Antonio M. Mora García, PhD in 2009 at the University of Granada (Spain), where he also got his Degree in Computer Sciences in 2001. Currently he is a Teaching Assistant at the Computer Architecture and Technology Department in the same university. He has participated in several funded researching projects, and published a number of papers in top-rated international conferences (such as GECCO, ALIFE, WCCI, PPSN or EVO*). His working areas include Ant Colony Optimization metaheuristic, Multi-Objective Optimization and Genetic Algorithms, and their applications to Pathfinding problems or Video Games among others.

Gustavo Romero López received the B.Sc. degree in Computer Science and the Ph.D. degree, both from the University of Granada, Spain. He has been working as programmer for CICYT proyect BIO96-0895. He is currently Lecturer at the Computer Architecture and Technology Department, University of Granada. His main interests in research are in neural networks and evolutionary computation.

Juan J. Merelo received the B.Sc. degree in theoretical physics and the Ph.D. degree, both from the University of Granada, Granada, Spain, in 1988 and 1994. He has been Visiting Researcher at Santa Fe Institute, Santa Fe, NM, Politecnico Torino, Turin, Italy, RISC-Linz, Linz, Austria, the University of Southern California, Los Angeles, and Université Paris-V, Paris, France. He is currently Full Professor at the Computer Architecture and Technology Department, University of Granada. His main interests include neural networks, genetic algorithms, and artificial life.