A service oriented evolutionary architecture: applications and results

Pablo García-Sánchez
Dept. of Computer Architecture
and Computer Technology
University of Granada, Spain
pgarcia@atc.ugr.es

ABSTRACT

This paper shows the stage of development of a Service Oriented Architecture for Evolutionary Algorithms and the first results obtained in two different areas. The abstract architecture is presented, with its assocciated implementation using a widely used technology. Results attained in experiments with parameter adaptation in distributed heterogeneous machines are presented and the usage of the architecture in Evolutionary Art is also applied.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; G.1.6 [**Mathematics of Computing**]: NUMERICAL ANALYSIS—*Optimization*

General Terms

Algorithms

Keywords

Service oriented architecture, framework, parameter setting, distributed algorithms, island model, evolutionary art

1. INTRODUCTION

Ian Foster defined in [6] the term *Service-Oriented Science*, that is, scientific research using interoperable and distributed networks of services, being the key of success the uniformity of interfaces, so researchers can discover and access to services without developing specific code for each data source, program or sensor. Therefore, this paradigm has the potential to increase scientific productivity thanks to the wide set of available distributed tools, and also increasing the automation of computation data analysis. On the other way, other trends such as Cloud Computing [3] or GRID [5] are leading to heterogeneous computational devices working at the same time. Moreover, many laboratories do not count with classic clusters, but the usual workstations used by scientists can behave in group as an heterogeneous cluster.

Nowadays, there are many frameworks for Evolutionary Algorithms (EAs), but all of them are Object Oriented statically programmed, not taking the advantages that Service Oriented Computing is able to offer. As services must be well-defined, encapsulated and reusable, it is necessary to abstract enough to have a good EA design. In [7] authors discuss about genericity in evolutionary computation software tools. Although their discussion is based in Object Oriented programming, the genericity of an EA framework can be applied to develop EC services, and extended with new functionalities.

Our previous work [8] presents an abstract Service Oriented Architecture for Evolutionary Algorithms (SOA-EA), describing the set of guidelines and steps to migrate from traditional development to SOA. It also presents a specific implementation, called OSGiLiath: an environment for the development of distributed algorithms, extensible via plugins architecture, and based on a wide-accepted software specification (OSGi: Open Services Gateway Initiative [12]).

This paper shows how this implementation has been used to obtain results in two different areas: one in the algorithmic scope (parameter tuning in heterogeneous devices [2]) and also in the application of EAs (Evolutionary Art [4]).

The rest of the work is structured as follows: after the state of the art, we present the developed service oriented architecture in Section 3. Then, the results of the first experiment in heterogeneous clusters are shown (Section 4), followed by an application in Evolutionary Art (Section 5). Finally conclusions and future lines of work are shown.

2. STATE OF THE ART

Even as Service Oriented Architecture is extensively used in software development, it is not widely extended in the EAs software scope. Firstly, there exist Object Oriented frameworks, such as Algorithm::Evolutionary, JCLEC or jMetal. Users implement specific interfaces of these frameworks (such as *individual* or *crossover*) and they group them in the source code. For example, creating an operator object that groups several operators. However, these frameworks are not compatible between them. For example, the operators created in JCLEC can not be used in jMetal (despite both are programmed in Java). Also, they can not control the services (operators) outside the source code. Parallelism and distribution are added in other frameworks, such as MALLBA, DREAM or ECJ, but using external libraries (such as MPI

GECCO'13, July 6-10, 2013, Amsterdam, The Netherlands.

or DRM), so the code that uses these libraries is mixed with the algorithm's code.

Even being distributed, these frameworks can not communicate with each other. HeuristicLab is one of the few plug-in and service oriented frameworks. It uses web services for communication, but just to distribute the load, after consulting a central database of available jobs. The work [13] contains a comparison and the references of the previous frameworks.

Other field related is the heterogeneous evolutionary computing, where two areas exist: heterogeneous hardware and heterogeneous parameters. In the first area authors study or adapt algorithms depending on the machine configuration [2, 11]. In the area of heterogeneous parameters, setting in each island (node) a different sets of parameters can also increase the performance of distributed EAs, as explained in [10, 15].

Our work presents a combination of these ideas, where a parameter tuning given by the computational power of the machines is performed. To our knowledge, there are not works that modify parameters of the EA depending of the node where the island is being executed.

3. A SERVICE ORIENTED ARCHITECTURE FOR EVOLUTIONARY ALGORITHMS

In [8] we presented an abstract architecture composed by loosely coupled, highly configurable and language-independent services for Evolutionary Computation (called SOA-EA). As example of implementation of this architecture, a complete process development using a specific service oriented technology (OSGi) was explained. With this implementation, less effort than classical development in integration, distribution mechanisms and execution time management has been attained. In addition, steps, ideas, advantages and disadvantages, and guidelines to create service oriented evolutionary algorithms were explained.

In [7], six criteria for qualify EA frameworks were presented: generic representation, fitness, operator, model, parameters management and configurable output. In our previous work we shown how SOA follows these lines of genericity, but can also extend them:

- Genericity in the service interfaces: service interfaces are established to create new implementations. Furthermore, these interfaces must be abstract enough to avoid their modification.
- Programming language independence: for example, services implemented in Java can use services implemented in C++ and vice-versa.
- Distribution transparency: it is not mandatory to use a specific library for the distribution, or modify the code to adapt the existing operators.
- Flexibility: easy to add and remove elements to use the self-adaptation or other mechanisms.

A specific implementation of our architecture (called OSGiLiath) has been developed using the OSGi service oriented technology, with a number of services already developed. These services can be combined in several ways to obtain different algorithms, and can be dynamically bound to change the needed EA features. In addition, new services can be added in execution time using our implementation. No specific source code for a basic distribution needs to be added, neither the existing source code has been modified to achieve the previous tasks.

4. ADAPTING THE POPULATION SIZE TO HARDWARE

One of the first experiments performed with OSGiLiath has been to establish the effect of the population size in homogeneous and heterogeneous clusters. The algorithm to be improved is a distributed Genetic Algorithm. The algorithm is steady-state: the offspring is mixed with the parents and the worst individuals are removed. A ring topology is used, and the best individual is sent after a fixed number of generations of each node (64). Two different parameter configurations have been used: 64 individuals per node (homogeneous size) and a different number of individuals proportional to the computational power of each node. An uniform crossover is used (with a rate of 0.5) and a bit-flip mutation (with a probability of 1/genome size).

The problems to evaluate are the Massively Multimodal Deceptive Problem (MMDP) [9] and the OneMax problem [16]. Each one requires different actions/abilities by the GA at the level of population sizing, individual selection and building-blocks mixing. The chromosome length is 150 for MMDP and 5000 for OneMax.

To test the algorithm two different computational systems have been used: an *heterogeneous cluster* and an *homogeneous cluster*. The first one is formed by 4 different computers of our lab with different processors, operating systems and memory size. The latter is a dedicated scientific cluster formed by homogeneous nodes. Table 4 shows the features of each system.

Acronyms for each configuration are HoSi (homogeneous population size), HeSi (heterogeneous population size), HoHa (homogeneous hardware) and HeHa (heterogeneous hardware).

Each different configuration has been tested 30 times. The number of individuals in each node of the HeSi configuration is proportional the computational power of each node. In this case the computational power has been calculated comparing the average number of generations obtained in the nodes of the HoSi/HeHa configuration for the MMDP problem. Thus, the HeSi configuration uses 98, 84, 66, and 8 individuals (from N1 to N4). Note that, having two nodes with the same processors and memory (N1 and N2), they have different computational power.

4.1 Results

The objectives of the parallel programming are to tackle large computational problems, increase the performance of algorithms in a finite time, or reduce time. In this work we focus in the last objective. As claimed by [1], the number of evaluations can be misleading in the parallel algorithms area. In our case, for example, the evaluation time is different in each node of the heterogeneous cluster, and the real algorithm speed could not be reflected correctly. However, the number of evaluations has been added in this section to better understand the results. Also, the total number of generations, and the maximum number of generations of the slower node are shown. It is difficult to compare between

Table 1: Details of the clusters used.

Name	Processor	Memory	Operating System	Network
Homogeneous cluster				
Cluster node	Intel(R) Xeon(R) CPU E5320 @ 1.86GHz	4GB	CentOS 6.7	Gigabit Ethernet
Heterogeneous cluster				
N1	Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz	4GB	Ubuntu 11.10 (64 bits)	Gigabit Ethernet
N2	Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz	4GB	Ubuntu 11.04 (64 bits)	Gigabit Ethernet
N3	AMD Phenom(tm) 9950 Quad-Core Processor @ 1.30Ghz	3GB	Ubuntu 10.10 (32 bits)	100MB Ethernet
N4	Intel (R) Pentium 3 @ 800MHz	768 MB	Ubuntu 10.10 (32 bits)	10MB Ethernet

HoHa and HeHa for the same reason: the evaluation time is different in each system (and also in each node).

4.1.1 MMDP Problem

Table 2 shows the results for the MMDP problem. In the HeHa system, adapting the population to the computational power of each node makes the algorithm to end significantly faster, but with the same number of evaluations (with no statistical significance). This can be explained because the evaluation time is different in all nodes. On the other hand, in the HoHa system, setting the same population sizes makes no difference in time and evaluations, that is, changing this parameter does not influence the performance of the algorithm (no statistically significance attained).

4.1.2 OneMax Problem

Results for OneMax are shown in Table 3. In this problem, adapting the population sizes decreases significantly the time for solving in the heterogeneous cluster, and, as before, the number of evaluations remains the same. In the homogeneous system, the effect of changing the population sizes is more evident, and this time the evaluations (and therefore, the time) are reduced (both significantly).

The efficiency on OneMax problems depends more on the ability to mix the building-blocks, and less on the genetic diversity and size of the population (as with MMDP). No genetic diversity is particularly required. When properly tuned, a simple Genetic Algorithm is able to solve OneMax in linear time. Sometimes, problems like OneMax are used as control functions, in order to check if very efficient algorithms on hard functions fail on easier functions.

5. COMPARING HISTOGRAMS IN EVOLUTIONARY ART

OSGiLiath has also been used to study the differences of using the information of the HSV (Hue, Saturation, Value) and RGB (Red, Green, Blue) histograms during the evolution of an aesthetical image. A service to access to Processing [14], a programming framework designed for visual artists, have been performed. In addition, services to measure the fitness, and implementations of individuals are also available in OSGiLiath. Processing is used inside the EA to model the individuals, generate their associate images and extract information of them (HSV, RGB and Average histograms) to fit with the histograms of a test image.

A steady-state evolutionary algorithm has been used. Each individual is randomly generated at the initialization of the EA. The genome size is 50 elements (circles of maximum radius of 128 pixels). Population size has been set to 32 individuals. Uniform crossover rate is 0.5, and a binary tournament has been chosen for selection. Mutation probability is 0.04 (the usual value of *1/genomesize*). Finally, the image

Table 4: Results for the different fitness. Only one histogram type is used, but the other values obtained are also added.

Differences used	Obtained RGB	Obtained HSV	Obtained Average
RGB	*0.267 ± 0.012*	0.170 ± 0.010	0.218 ± 0.009
HSV	0.227 ± 0.017	*0.265 ± 0.021*	0.246 ± 0.010
Average	0.173 ± 0.012	0.294 ± 0.013	*0.234 ± 0.010*

size for each individual is 256x256 pixels. The individuals have been compared with the histograms obtained from an aesthetic predefined image to guide the evolution.

Three different fitness functions using color histogram have been tested and added to OSGiLiath as services: difference between the HSV and RGB histograms, and an average difference of the two histograms at the same time. Table 4 show the attained results. Experiments show that better results in terms of similarity are obtained using the HSV comparison (due to the noisy information provided by the RGB). This is a basic image metric, only used by purposes of proof-of-concept and more complex measurements will be studied in future works.

6. CONCLUSIONS

Service Oriented Computing is a new trend where computational resources cooperate in an automatic way without taking into account programming language or operating system. Also, other trends, such as Cloud Computing are providing a massively amount of heterogeneous computational devices. This has been the motivation to develop SOA-EA and OSGiLiath.

The first applications have been a preliminary study about adapting the population size of an EA to computational power of different nodes in an heterogeneous cluster. Results show that adapting the population size decrease the execution time significantly in heterogeneous clusters, while changing this parameter in homogeneous clusters not always performs better. This is a promising start for adapting EAs to the computational power of each execution node.

As future work a scalability study will be performed, with more computational nodes and larger problem instances. Moreover, other parameters such as migration rate or crossover probability will be adapted to the execution nodes. This studies will lead to automatic adaptation during runtime, with different nodes entering or exiting in the topology during the algorithm execution or adapting to the current load of the system.

Finally, new experiments in the field of Evolutionary Art will be performed.

The project development is explained and also avaible for download and modification under a GPL V3 License at `http://www.osgiliath.org`

Table 2: Results for the MMDP problem.

Configuration	Max. generations	Total generations	Total evaluations	Time (ms)
HoSi/HeHa	146401,48 ± 65699,69	380967,25 ± 168568,84	24382416,51 ± 10788405,87	136914,03 ± 60028,48
HeSi/HeHa	96051,5 ± 45110,90	289282,3 ± 135038,10	21784528,66 ± 10161989,38	109875,76 ± 49185,51
HoSi/HoHa	107334,46 ± 78167,19	393119,86 ± 241835,27	25273201,06 ± 15386663,12	237759,43 ± 178709,86
HeSi/HoHa	149732,6 ± 81983,74	438171,16 ± 240169,19	24430043,46 ± 13395037,34	245776,93 ± 134715,52

Table 3: Results for the OneMax problem.

Configuration	Max. generations	Total generations	Total evaluations	Time (ms)
HoSi/HeHa	4739,41± 305,32	12081,51± 776,35	773729,03± 49686,72	72152,32± 4994,71
HeSi/HeHa	3438,03 ± 149,47	11277,33± 471,77	794157,73± 31843,10	61870,2 ± 2518,74
HoSi/HoHa	3133,36± 101,70	12347,83± 394,99	790773,33± 25279,52	62105,03± 1964,75
HeSi/HoHa	13897,86± 625,27	20725,63± 929,43	651952,8 ± 29114,54	56120,53± 2491,92

7. ACKNOWLEDGMENTS

This work has been supported in part by FPU research grant AP2009-2942 and projects EvOrq (P08-TIC-03903), UGR PR-PP2011-5 and TIN2011-28627-C04-02.

8. REFERENCES

[1] E. Alba and G. Luque. Evaluation of parallel metaheuristics. In Springer, editor, *Parallel Problem Solving from Nature (PPSN)*, volume 4193 of *LNCS*, pages 9–14, 2006.

[2] Enrique Alba, Antonio J. Nebro, and José M. Troya. Heterogeneous computing and parallel genetic algorithms. *Journal of Parallel and Distributed Computing*, 62(9):1362 – 1385, 2002.

[3] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25:599–616, June 2009.

[4] David W Corne and Peter J Bentley. *Creative evolutionary systems*. Morgan Kaufmann, 2001.

[5] Simon J. Cox, Matt J. Fairman, Gang Xue, Jasmin L. Wason, and Andy J. Keane. The grid: Computational and data resource sharing in engineering optimisation and design search. In *30th International Workshops on Parallel Processing (ICPP 2001 Workshops), 3-7 September 2001, Valencia, Spain*, pages 207–212. IEEE Computer Society, 2001.

[6] I. Foster. Service-oriented science. *Science*, 308(5723):814, 2005.

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

[8] P. García-Sánchez, J. González, P.A. Castillo, M.G. Arenas, and J.J. Merelo-Guervós. Service oriented evolutionary algorithms. *Soft Computing*, pages 1–17, 2013. 10.1007/s00500-013-0999-5.

[9] David E. Goldberg, Kalyanmoy Deb, and Jeffrey Horn. Massive multimodality, deception, and genetic algorithms. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature, 2*, pages 37–48, Amsterdam, 1992. Elsevier Science Publishers, B. V.

[10] Yiyuan Gong and Alex Fukunaga. Distributed island-model genetic algorithms using heterogeneous parameter settings. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2011, New Orleans, LA, USA, 5-8 June, 2011*, pages 820–827. IEEE, 2011.

[11] Yiyuan Gong, Morikazu Nakamura, and Shiro Tamaki. Parallel genetic algorithms on line topology of heterogeneous computing resources. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 1447–1454, New York, NY, USA, 2005. ACM.

[12] OSGi Alliance. OSGi service platform release 4.2, 2010. Available at: `http://www.osgi.org/Release4/Download`.

[13] José Parejo, Antonio Ruiz-Cortés, Sebastián Lozano, and Pablo Fernandez. Metaheuristic optimization frameworks: a survey and benchmarking. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 16:527–561, 2012. 10.1007/s00500-011-0754-8.

[14] C. Reas and B. Fry. *Processing: A Programming Handbook for Visual Designers and Artists*. Working paper series (National Bureau of Economic Research). Mit Press, 2007.

[15] Carolina Salto and Enrique Alba. Designing heterogeneous distributed gas by efficiently self-adapting the migration period. *Applied Intelligence*, 36:800–808, 2012.

[16] J.D. Schaffer and L.J. Eshelman. On Crossover as an Evolutionary Viable Strategy. In R.K. Belew and L.B. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 61–68. Morgan Kaufmann, 1991.