Using pool-based evolutionary algorithms for scalable and asynchronous distributed computing

J.J. Merelo[1], A.M. Mora[1], C. M. Fernandes[1], M. G. Arenas[1], and Anna I. Esparcia-Alcázar[2]

[1] University of Granada
Department of Computer Architecture and Technology, ETSIIT
18071 - Granada
{jmerelo,amorag,cfernandes,mgarenas}@geneura.ugr.es
[2] S2 Grupo
aesparcia@s2grupo.es

1 Introduction

Many studies on evolutionary algorithms (EAs) rely on traditional execution environments with single memory and CPU, possibly extended to parallel and even distributed environments, provided that there are certain conditions, such as synchrony, homogeneity and centralized operations. However, in the last few years the range of possible computational environments has been extended greatly, to the point that it is possible to achieve a bigger computational raw performance by creating ad hoc, loosely linked, and heterogeneous frameworks where EAs can be run. One of such targets are the so-called *volunteer computing* or *desktop grid* environments, which have been used extensively so far in evolutionary algorithms, for instance in [1].

Pool-based systems introduce a repository of solutions or *pool*, which can be accessed in a distributed way, and in which all the population or a part of it can be deposited. This supports a wide range of possible models, from a classical and asynchronous island model in which the pool is used by all nodes in lockstep to interchange individuals, to a data-flow based system in which nodes pick up individuals from the pool and deposit results of the evolution back. It can even support all of them at the same time. Since there is the possibility of decoupling the population from the operations applied on them, pool EAs open the way for completely novel evolutionary algorithm models.

Besides, several implementations are also possible: relational database systems or object stores, as well as file synchronization services such as Dropbox or SugarSync. In this proposal for the workshop we will describe our work on this kind of systems, its advantages and disadvantages, and what we can expect from them. In any of them, pool EAs can easily scale to any number of nodes, since new ones can be added without any special provision on the server or giving notice to existing nodes; they can also leave by just stopping sending requests to the server, which makes them amenable for volunteer computing as said above.

2 State of the Art

In this section we will examine pool-based distributed computing systems, mainly those that have been applied to evolutionary algorithms. The most popular model for asynchronous distributed algorithms is called *A-teams*, where A stands for *asynchronous* [2]. A-Teams combine different algorithms that share a memory in closed loops and are a way of specifying data flow among different methods to solve a problem. A-Teams are not intrinsically evolutionary methods but have been successfully applied in the last decades to a wide variety of problems [3]; their authors have released a toolkit that can be used to implement solutions to different problems. A-Teams can be implemented in many different ways, but they often refer to a *pool* or shared memory from which solutions (or sets of them) can be drawn, improved and put back, or to where newly constructed solutions can be shared among all the agents participating in the experiment.

Taking then one step down and entering the realm of the implementations (away from the models exposed above), several authors have directly implemented evolutionary algorithms in a pool based architecture, where the basic idea is to use a (more or less persistent) store of solutions from which the evolutionary algorithm draws its individuals, instead of having the population as a data structure that is taken from one method to the next. The first papers in the 90s used shared memory systems such as Linda [4]. Lately, multi-threaded systems with a shared memory [5] have been proposed; this memory can be read from all threads, but is divided in chunks writable by only one of the threads. Relational database systems [6] have also been used, proving their capability for avoiding algorithms with explicit synchronization and their fault-tolerance, at least to client failure, providing a persistent storage for population from which solutions can be, later on, retrieved. A database is, for instance, used in Distributed BEAGLE [7], which separates evolution and evaluation with a single *evolver* client independent from the *evaluator* clients, both working with a central database.

Even if the database is a single point of failure, this can be avoided by replication; besides, the state of evolution is partially held by anyone of the clients at a particular moment, so even in the event of a database failure all the information is not lost.

3 Pool based EAs: Models

Being the basic operations in pool-based algorithms CRUD (create, read, update and delete) functions, we have been working in two different types of models

- Pool contains the whole population, nodes are used for most operations. In this type of algorithms, every node, using some criterium (such as plain availability of individuals ready for processing, or selection of the best) reads a set of individuals, applies genetic operators, evaluation, or a single generation to them and deposits the whole result in the pool, which is thus incremented in each step, and until the end of the algorithm. Either the same node or

another must monitor the population to maintain a certain selective pressure and keep the size in check. This was used in SofEA [8].
– Pool is simply used as a migration conveyor in an island-based evolutionary algorithm. Evolutionary algorithms are run independently en each node and these using different migration and incorporation policies, send some individuals to the pool and pick up from it others to be incorporated (or not, according to policy) into the population. This approach was used in [9–11].

Both approaches have its merit: island-based models, since the amount of operations on the pool are minimal, is scalable to a great amount of nodes; on the other hand, using nodes for a single operation (of whatever complexity) is more fault tolerant, since the state of the algorithm is in several nodes at the same time.

4 Pool based EAs: Implementations

As mentioned above, any system that supports CRUD operations can be used to implement a pool. However, the systems used by us so far fall into two broad categories:

– Object stores, be them relational database systems [12] or NoSQL systems [13]. In this case all nodes perform their operations remotely, using the system programming interface.
– File synchronization systems (used in [11]) which maintain directories with the same content. Nodes write individuals into these files or in particular files within those directories; all communication is done through the directory that is shared across all nodes.

The main advantage of maintaining a pool in a shared directory is the locality of operations and its simplicity: writing a file is all that is needed, and it is done at the speed of the local filesystem (which can be sped up as much as possible). However, these commercial systems have their own synchronization schedule, and synchronization is not usually immediate, which means systems do not scale well unless experiments take a long time. Object stores, on the other hand, have as their main disadvantage its latency; unless operations in the nodes are significantly faster than the round-trip from node to pool, there is little advantage in using them.

5 Conclusions

Pool-based EAs are a challenge from several points of view: mapping the traditional evaluation/reproduction/selection loop to a new architecture, and doing so in a way that does not hinder scalability and then mapping this new algorithm to a framework that is suitable for it. In this paper we have made an overview of our work (and other's) in this area, and expect to compare this kind of solutions with other kind of distributed and parallel evolutionary algorithms.

Acknowledgments

This work is supported by project TIN2011-28627-C04-02 awarded by the Spanish Ministry of Science and Innovation and P08-TIC-03903 awarded by the Andalusian Regional Government.

References

1. Gonzalez, D., de Vega, F., Trujillo, L., Olague, G., Araujo, L., Castillo, P., Merelo, J., Sharman, K.: Increasing GP computing power for free via desktop GRID computing and virtualization. In: Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on. (Feb. 2009) 419–423
2. Talukdar, S., Murthy, S., Akkiraju, R.: Asynchronous teams. International Series in Operations Research and Management Science (2003) 537–556
3. Jedrzejowicz, P.: A-teams and their applications. In Nguyen, N., Kowalczyk, R., Chen, S.M., eds.: Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems. Volume 5796 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2009) 36–50
4. Davis, M., Liu, L., Elias, J.: VLSI circuit synthesis using a parallel genetic algorithm. In: Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on. (jun 1994) 104 –109 vol.1
5. Roy, G., Lee, H., Welch, J., Zhao, Y., Pandey, V., Thurston, D.: A distributed pool architecture for genetic algorithms. In: Evolutionary Computation, 2009. CEC '09. IEEE Congress on. (May 2009) 1177–1184
6. Bollini, A., Piastra, M.: Distributed and persistent evolutionary algorithms: a design pattern. In: Genetic Programming, Proceedings EuroGP´99. Number 1598 in Lecture notes in computer science, Springer (1999) 173–183
7. Gagné, C., Parizeau, M., Dubreuil, M.: Distributed BEAGLE: An environment for parallel and distributed evolutionary computations. In: Proc. of the 17th Annual International Symposium on High Performance Computing Systems and Applications (HPCS). Volume 2003. (2003) 201–208
8. Merelo Guervos, J.J., Mora, A.M., Cruz Almaguer, J.A., Esparcia-Alcazar, A.I., Cotta, C.: Scaling in distributed evolutionary algorithms with persistent population. In: CEC 2012 Proceedings. (2012) Accepted, to be published.
9. Merelo, J.J., Castillo, P., Laredo, J., Mora, A., Prieto, A.: Asynchronous distributed genetic algorithms with Javascript and JSON. In: WCCI 2008 Proceedings, IEEE Press (2008) 1372–1379
10. Merelo, J.J.: Fluid evolutionary algorithms. In: IEEE Congress on Evolutionary Computation, IEEE (2010) 1–8
11. Arenas, M.G., Guervós, J.J.M., Castillo, P.A., Laredo, J.L.J., Romero, G., Mora, A.M.: Using free cloud storage services for distributed evolutionary algorithms. In Krasnogor, N., Lanzi, P.L., eds.: GECCO, ACM (2011) 1603–1610
12. Merelo, J.J., García, A.M., Laredo, J.L.J., Lupión, J., Tricas, F.: Browser-based distributed evolutionary computation: performance and scaling behavior. In: GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation, New York, NY, USA, ACM Press (2007) 2851–2858
13. Merelo-Guervós, J.J., Mora, A., Cruz, J.A., Esparcia, A.I.: Pool-based distributed evolutionary algorithms using an object database. In di Chio et al., C., ed.: EvoApplications 2012 Proceedings. (2012) 441–450