Influence of population size in distributed Evolutionary Algorithms in heterogeneous clusters

Pablo García-Sánchez[a], Jesús González[a], María Isabel G. Arenas[a], Pedro A. Castillo[a], Carlos Fernandes[b], Antonio Miguel Mora[a], Juan Julián Merelo[a]

[a]*Department of Computer Architecture and Computer Technology and CITIC-UGR, University of Granada, Granada, Spain. Tel: +34958241778. Fax: +34958248993*
[b]*LaSEEB-ISR-IST, Technical University of Lisbon (IST), Lisbon, Portugal*

Abstract

This paper presents a study on population size adaptation in a distributed Evolutionary Algorithm executed on a heterogeneous cluster. The total number of individuals is divided taking into account the computational power of each node of a heterogeneous network of computers. Results show that setting the population size according to the computational power in the heterogeneous cluster decreases the time required to obtain the optimum in two problems with different characteristics and computational demands (MMDP and OneMax), while the same parameter configuration could not improve the time in a homogeneous cluster. Also, a study of the influence of the different population sizes on each stage of the algorithm is presented. This opens a new research line on the adaptation (offline or online) of parameters to the computational power of the devices.

Keywords: Evolutionary Algorithms, Genetic Algorithms, Heterogeneous computation, Distributed computing

1. Introduction

Evolutionary Algorithms (EAs) are a general technique for solving optimization and search problems based on the evolution of species and natural selection. These algorithms are formed by a population of possible solutions (called *individuals*) that compete using their *fitness* (quality of adaptation) with the rest of solutions. In each iteration of the algorithm (or *generation*) the population evolves by means of selection and recombination/mutation to create a new set of candidates, until a *stop criterion* (i.e. number of generations) is met. Fitness function is a quality function that gives the grade of adaptation of an individual respect the others. This function describes the problem to solve.

New trends in distributed computing such as Cloud Computing [5], GRID [3] or Service Oriented Science [9] are leading to heterogeneous computational devices, for instance laptops, tablets or desktop PCs, working in the same environment. Thus, many laboratories, which do

Email addresses: pgarcia@atc.ugr.es (Pablo García-Sánchez), jesus@atc.ugr.es (Jesús González), maribel@ugr.es (María Isabel G. Arenas), pedro@atc.ugr.es (Pedro A. Castillo), cfernandes@laseeb.org (Carlos Fernandes), amorag@geneura.ugr.es (Antonio Miguel Mora), jmerelo@geneura.ugr.es (Juan Julián Merelo)

not count with classic clusters but the usual workstations used by scientists, can leverage this motley set as a heterogeneous cluster. Distributed Evolutionary Algorithms (dEAs) [4, 15] have been tested successfully in these systems and they are very popular because their implementation is not complicated and they exploit a coarse grained parallelism with sporadic communications, being fit to be executed in distributed architectures such as clusters or GRIDs [16].

In distributed EAs a number of nodes executes simultaneously the EA, working with different sub-populations (or islands) at the same time. Each certain number of generations one or more individuals are interchanged (migrated) between populations. Figure 1 shows this model with a ring topology.

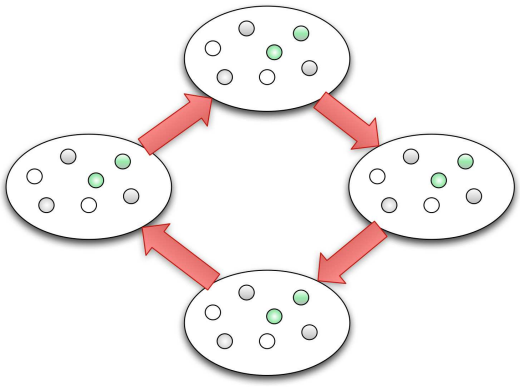


Figure 1: Island model using a ring topology for migration of individuals.

Distributed EAs can be executed in homogeneous clusters with the same parameters in all nodes (homogeneous dEAs) or with differences in the parameters or nodes (heterogeneous dEAs). It has also been showed [2] that dEAs with the same parameters are even more efficient in heterogeneous hardware configurations than in homogeneous devices. This can be explained by different reasons, such as different memory access times, cache sizes, or even implementation languages or compilers in each machine, leading to a different exploitation/exploration rate of the search space. Heterogeneous parameters configuration has also been showed more efficient time-wise than a fixed set of parameters for different problems [13].

These proofs have motivated us to study a combination of both ideas in this paper: dEAs in a heterogeneous set of nodes with differents parameters adapted to each node. In this study, the parameter to adapt to the computational power of each node has been the population size of each island.

In this work, a heterogeneous distributed system has been used to give an insight to the following questions:

- Can a distributed EA be adapted to leverage the capability of a heterogeneous cluster?
- What is the effect of adapting the population size to the computational power, as proposed in this paper?
- Is there any difference between using the same population sizes in a homogeneous or a heterogeneous cluster?

- How is each stage of the algorithm (selection, migration...) affected by the different configurations?

The rest of the work is structured as follows: after a presentation of the state of the art in the parameter adaptation and load-balancing in dEAs, we present the developed algorithms and experimental setting (Section 3). Then, the results of the experiments are shown (Section 4), followed by conclusions and suggestions for future work lines.

2. State of the art

In the field of Evolutionary Computation (EC) there are two different approaches about the algorithm parameter setting: *parameter control* and *parameter tuning* [8]. The first one refers to setting up a number of parameters of an Evolutionary Algorithm (EA) and changing these parameters in running time (online). The parameter tuning consists in establishing a good set of parameters before the run (offline), and do not change them during runtime.

Computational performance of nodes or network speed can also be inherent parameters of an algorithm. In [2] Alba et al. compared a distributed Genetic Algorithm (dGA), one of the sub-types of EAs, in homogeneous and heterogeneous clusters. Super-linear performance was obtained in the heterogeneous ones, being more efficient than the same algorithm running in homogeneous machines, but the parameter set was the same in both clusters. Some authors have expanded this idea by adapting the algorithm to be executed: Domínguez et al. [7] presented a distributed hybrid meta-heuristic that combines two different EAs: Genetic Algorithms (GAs) and Simulated Annealing (SA). Their system executes the heavy (in computational terms) algorithms (GAs) in faster nodes, and simpler meta-heuristics (SA) in slower nodes, obtaining better results than other configurations. As before, the parameters were not adapted to the node. Gong et al. in [14] studied different configurations of heterogeneous machines for a tree topology. However, the heterogeneity was simulated in a homogeneous cluster with programs to add computational load. Load-balancing was also applied taking into account the computational load of the nodes in the work of Garamendi et al. [10]: a small benchmark was executed in all nodes at the beginning of the algorithm in order to distribute individuals of an Evolutionary Strategy (ES). However, there was no communication between the nodes and the algorithm parameters were not adapted.

In the area of heterogeneous parameters, but homogeneous hardware, setting a random set of parameters in each node can also increase the performance of a distributed Genetic Algorithm, as explained by Gong and Fukunaga in [13]. That model outperformed a tuned canonical dGA with the same parameter values in all islands. Finally, adapting the migration rate produced better results than homogeneous periods, as explained by Salto and Alba in [17]. Previous works were only tested in homogeneous clusters.

Our work presents a combination of some of the previous ideas: some parameters were adapted to the computational power of each machine used, and tested them in different hardware systems. To our knowledge, there are not works that modify parameters of the EA depending of the node where the island is being executed.

3. Experimental setup

This section presents the parameters and platforms to conduce the experiments.

3.1. Algorithm used

The experimentation is centered in a distributed GA. Figure 2 shows the pseudo-code of the used algorithm. The algorithm is steady-state, i.e. the offspring is mixed with the parents and the worst individuals are removed. The used neighborhood topology for migration between islands (nodes) is a ring (see Figure 1). The best individual is sent to the neighbour in the ring, after a fixed number of generations in each island. The algorithm stops when the optimum (the solution to the problem) is found.

```
population ← initializePopulation()
while stop criterion not met do
  parents ← selection(population)
  offspring ← recombination(parents)
  offspring ← mutation(offspring)
  population ← population + offspring
  if time to migrate then
    migrants ← selectMigrants(population)
    remoteBuffer.send(migrants)
  end if
  if localBuffer.size ≠ zero then
    population ← population + localBuffer.read()
  end if
  population ← removeWorst(population)
end while
```

Figure 2: Pseudo-code of the used dEA: a distributed Genetic Algorithm (dGA).

3.2. Problems

The problems to evaluate are the Massively Multimodal Deceptive Problem (MMDP) [12] and the OneMax problem [18]. Each one requires different actions/abilities by the GA at the level of population sizing, individual selection and building-blocks mixing. The MMDP is designed to be difficult for an EA, due to its multimodality and deceptiveness. Deceptive problems are functions where low-order building-blocks do not combine to form higher order building-blocks. Instead, low-order building-blocks may mislead the search towards local optima, thus challenging search mechanisms. MMDP it is composed of k subproblems of 6 bits each one (s_i). Depending of the number of ones (unitation) s_i takes the values shown in Table 1.

The fitness value is defined as the sum of the s_i subproblems with an optimum of k (Equation 1). The search space is composed of 2^{6k} combinations from which there are only 2^k global solutions with 22^k deceptive attractors. Hence, a search method have to find a global solution out of 2^{5k} additionally to deceptiveness. In this work $k = 25$.

$$f_{MMDP}(\vec{s}) = \sum_{i=1}^{k} fitness_{s_i} \tag{1}$$

OneMax is a simple linear problem that consists in maximising the number of ones in a binary string. That is, maximize the expression:

$$f_{OneMax}(\vec{x}) = \sum_{i=1}^{N} x_i \tag{2}$$

Table 1: Basic deceptive bipolar function (s_i) for MMDP.

Unitation	Subfunction value
0	1.000000
1	0.000000
2	0.360384
3	0.640576
4	0.360384
5	0.000000
6	1.000000

Table 2: Details of the clusters used.

Name	Processor	Memory	Operating System	Network
Homogeneous cluster				
HoN[1-4]	Intel(R) Xeon(R) CPU E5320 @ 1.86GHz	4GB	CentOS 6.7	Gigabit Ethernet
Heterogeneous cluster				
HeN1	Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz	4GB	Ubuntu 11.10 (64 bits)	Gigabit Ethernet
HeN2	Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz	4GB	Ubuntu 11.04 (64 bits)	Gigabit Ethernet
HeN3	AMD Phenom(tm) 9950 Quad-Core Processor @ 1.30Ghz	3GB	Ubuntu 10.10 (32 bits)	100MB Ethernet
HeN4	Intel (R) Pentium 3 @ 800MHz	768 MB	Ubuntu 10.10 (32 bits)	10MB Ethernet

3.3. Hardware and parameter configurations

Four configurations have been tested:

- HoSi/HoHa: Homogeneous Size/Homogeneous Hardware. The same population size in each island in a homogeneous cluster.
- HoSi/HeHa: Homogeneous Size/Heterogeneous Hardware. The same population size in each island in a heterogeneous cluster.
- HeSi/HoHa: Heterogeneous Size/Homogeneous Hardware. Different population sizes in each island in a homogeneous cluster.
- HeSi/HeHa: Heterogeneous Size/Heterogeneous Hardware. Different population sizes in each island in a heterogeneous cluster.

Two different computational systems have been used: an *heterogeneous cluster* and an *homogeneous cluster*. The first one is formed by four 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 2 shows the features of each system and the name of the nodes.

3.4. Homogeneous Size configuration

In this configuration, each node have 256 individuals (so, the total number is 1024). After executing the algorithm 40 times for problem in the heterogeneous cluster, we have obtained the average number of generations in each node, as can be seen in Table 3. Note how the generations attained (and their proportion in each node) to reach the optimum depends of the problem used (besides the hardware).

Table 3: Average number of generations to finish in each node of the heterogeneous cluster with heterogeneous size.

Node	HeN1	HeN2	HeN3	HeN4
MMDP problem				
Generations	10990,25	10732,075	7721,15	717,95
Proportion	36,43	35,58	25,59	2,38
OneMax problem				
Generations	2430,27	2353,77	1423,77	91,5
Proportion	38,58	37,36	22,6	1,45

3.5. Heterogeneous Size configuration

Our hypothesis consist in validating the following hypothesis: adapting the population size to the computational power of the nodes of a heterogeneous cluster presents an improvement in execution time. In this work, we have used the average number of generations obtained in the HoSi/HeHa configuration for both problems to determine the computational power of the heterogeneous machines. This comparison takes into account all the evolutionary process in a fair manner (proportional to the memory, processor and network usage), instead a traditional benchmark that usually relies only the CPU speed. Although this is not obviously the best way, it is a possible way to establish the computational power for the experiments of this work and to determine if changing the population size according the computational power reduces the time of the whole system. It should be considered that the contribution of this work is not the way we have used to calculate these sizes, but compare the algorithm with parameters adapted to their power.

Thus, we have used the obtained average number of generations in the previous sub-section to set proportionally the sizes in the HeSi/HeHa and HeSi/HoHa configurations, dividing the total number of individuals (1024). Note that, having two nodes with the same processors and memory (HeN1 and HeN2), they have different computational power: this might be produced by different operating systems, virtual machine versions, or number of processes being executed.

Table 4 summarizes all the parameters used in the experiments.

3.6. Framework

In order to deal with the operating system and architecture heterogeneity, the OSGiLiath framework [11], based in Java, has been used in this work. This is a service-oriented evolutionary framework that automatically configures the services to be used in a local network. In this case, each node offers a migration buffer to accept foreign individuals. Also, in order to reduce bottlenecks in distributed executions, asynchronous communication has been provided to avoid idle time using reception buffers (that is, the algorithm does not wait until new individuals arrive, but the buffers cannot be used again until the reception is done). This kind of communication offers an excellent performance when working with different nodes and operating systems, as demonstrated in [2]. The transmission mechanism is based on ECF Generic server (over TCP)[1]. The source code of the algorithms used in this work is available in `http://www.osgiliath.org` under a LGPL V3 License.

[1] `http://www.eclipse.org/ecf/`

Table 4: Parameters used in all configurations.

Name	Value
Crossover type	Uniform crossover
Crossover rate	0.5
Mutation rate	1/individual size
Selection	2-tournament
Replacement	Steady-state
Generations to migrate	64
Individuals to migrate	1
Stop criterion	Optimum found
Individual size for MMDP	150
Individual size for OneMax	5000
Runs per configuration	40
Total individuals	1024
Population size in each node in HoSi	256
Population sizes in HeSi for MMDP	374, 364, 262 and 24 (from N1 to N4)
Population sizes in HeSi for OneMax	396, 382, 232 and 14 (from N1 to N4)

4. Results

The main objetives of parallel programming are to tackle large computational problems, increase the performance of algorithms in a finite time, or reduce computational time to solve the problem (reaching the optimum). In this work, we focus in the last objective. As claimed by Alba and Luque in [1], assessing the performance of a parallel EA by the number of function evaluations required to attain a solution may be misleading. In our case, for example, the evaluation time is different in each node of the heterogeneous cluster, so the real algorithm speed could not be reflected correctly. However, the number of evaluations has been included in this section to better understand the results. The total number of generations, and the maximum number of generations required by the faster node in each configuration are also shown. It is difficult to compare the performance of HoHa and HeHa for the same reason: the evaluation time is different in each system (and even in each node). That is, in this work, we are not making the heterogeneous cluster comparable or better than the homogeneous one (because they are, obviously, different).

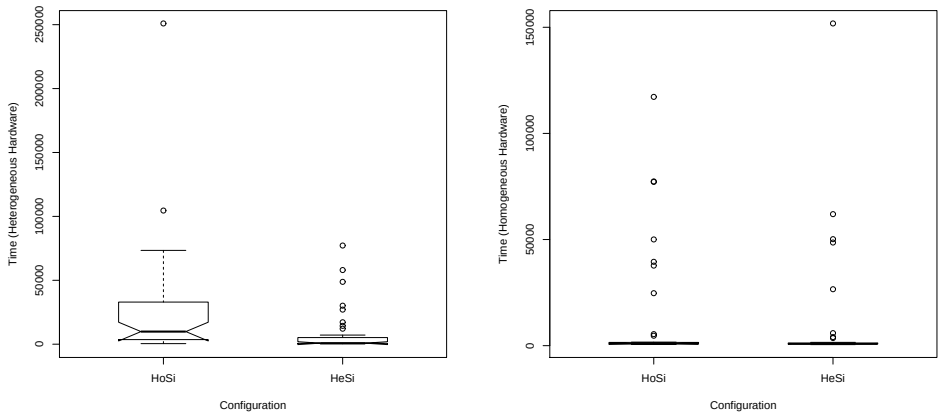
4.1. MMDP results

Table 5 shows the results for the MMDP problem. These results are also shown in the boxplots of Figure 3 (time) and Figure 4 (evaluations). Table 7 shows the statistical significance of the results. First, a Kolmogorov-Smirnov test is performed to assess the normality of the distributions. If the results fit a normal distribution, then a Student's T-Test is calculated. Otherwise, the non-parametric test Wilcoxon test is applied (see [6] for a tutorial for comparing EAs).

In the HeHa system, adapting the population to the computational power of each node makes the algorithm finish significantly earlier, and also, needing a lower number of evaluations to reach the solution. 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 has no influence in the algorithm's performance (p-value=0.52 and 0.08).

Table 5: Results for the MMDP problem.

Configuration	Max. generations	Total generations	Total evaluations	Time (ms)
HoSi/HeHa	11194,8 ± 18810,08	30161,42 ± 50722,03	7723372,8 ± 12984841,71	27871,075 ± 44583,14
HeSi/HeHa	2506,1 ±5308,872	8683,9 ± 18459,58	2453677 ±5217896,18	8110,9 ± 17162,86
HoSi/HoHa	2614 ±5889,93	10259,22 ± 23153,23	2628409,6 ± 5927278,22	11560,8 ± 26072,14
HeSi/HoHa	5411,92 ±15608,81	10689,15 ± 30790,7	1844908,1 ± 5314771,88	9520,325 ± 27237,35



(a) Heterogeneous cluster

(b) Homogeneous cluster

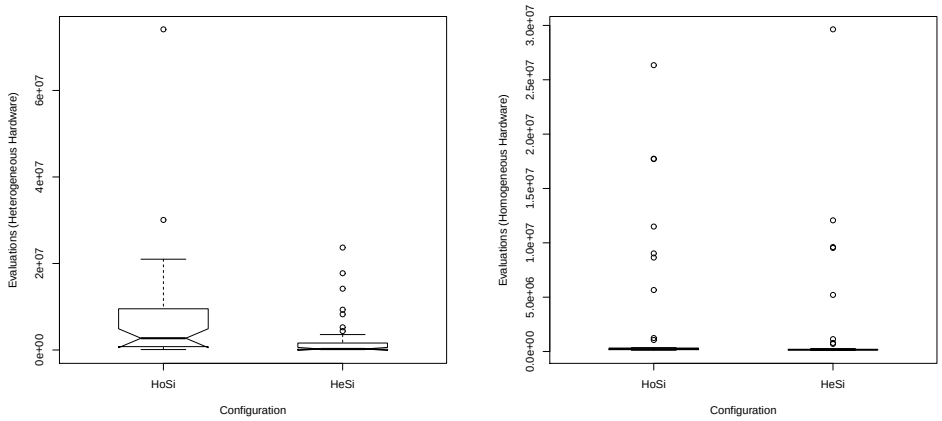
Figure 3: Time to obtain the optimum in the MMDP problem (milliseconds).

To see the difference of how the evolution is being performed, the average fitness in each node of HeHa is shown in Figures 5 and 6. As it can be seen, with the HeSi (Figure 6), the local optima are overtaken in less time than HoSi (Figure 5). This can be explained because in HeSi, the migration from HeN4 to HeN1 is performed faster, adding more heterogeneity to the whole system. Gaps in the figures correspond to the time while the nodes are sending the migrant individual to other nodes (not while they are receiving them). In the HoHa systems, the populations are evolved at the same time, being the average fitness similar in all nodes during all run.

4.2. OneMax results

Results for this problem are shown in Table 6 and Figures 7 and 8. In this case, adapting the population sizes significantly decreases the running time for solving in the heterogeneous cluster, but in this case, the number of evaluations is increased (see statistical significance in Table 7). In the homogeneous system, the effect of changing the population sizes is clearer, and this time the number of evaluations (and therefore, the time) are reduced (both significantly).

The efficiency on OneMax problem depends mainly 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 ones. As it can be seen in



(a) Heterogeneous cluster

(b) Homogeneous cluster

Figure 4: Number of evaluations for MMDP problem.

Table 6: Results for the OneMax problem.

Configuration	Max. generations	Total generations	Total evaluations	Time (ms)
HoSi/HeHa	2430,34 ± 70,16	6299,31 ± 250,87	1614673,45 ± 64223,09	160713,65 ± 8873,46
HeSi/HeHa	2643,34 ±150,82	7969,58 ±214,92	1802321,65 ± 30511,96	151822,75 ±4764,95
HoSi/HoHa	1791,32 ± 31,64	7111,05 ±125,11	1822476,8 ±32029,78	141176,1 ±2493,72
HeSi/HoHa	13698,12 ± 406,85	16012,625 ± 482,61	895698,2 ± 29520,99	77898,85 ± 2935,57

Figure 9, the average fitness of all populations are increasing in linear way in the HoSi/HeHa configuration. However, the slower node evaluates extremely fewer times. On the other side, in Figure 10, smaller population sizes make that slower nodes increase the number of evaluations, but the average fitness is also maintained in linear way (and in smaller increase rate) between migrations. However, the other nodes still perform a higher number of evaluations. That is the reason why the number of evaluations is higher in HeHa, and lower in HoHa. Computational time is more efficiently spent in faster nodes, having a higher chance to cross the individuals. In addition, due to the larger size of individuals in the OneMax problem (5000 bits vs. 150 of the MMDP), the transmission time is larger, (white gaps in the figures). It also implies that HeN4 sends its best individual to HeN1 in an extremely large amount of time when using HoSi (every 64 generations).

4.3. Running time analysis

This sub-section analyses the time spent by each node of the clusters in every stage of the EA for each configuration. Tables 8 and 9 show the average and standard deviation of the time spent in each stage of the algorithm (He=Heterogeneous cluster, Ho=Homogeneous cluster). Figures 11 and 12 graphically compare these results. As it can be seen, the migration is the most time consuming operation in all configurations, being the migration in HeHa more expensive than in HoHa. This happens because we are using the multi-purpose laboratory network to communicate the nodes, instead of the specific one used in the HoHa system. Note that the standard deviation

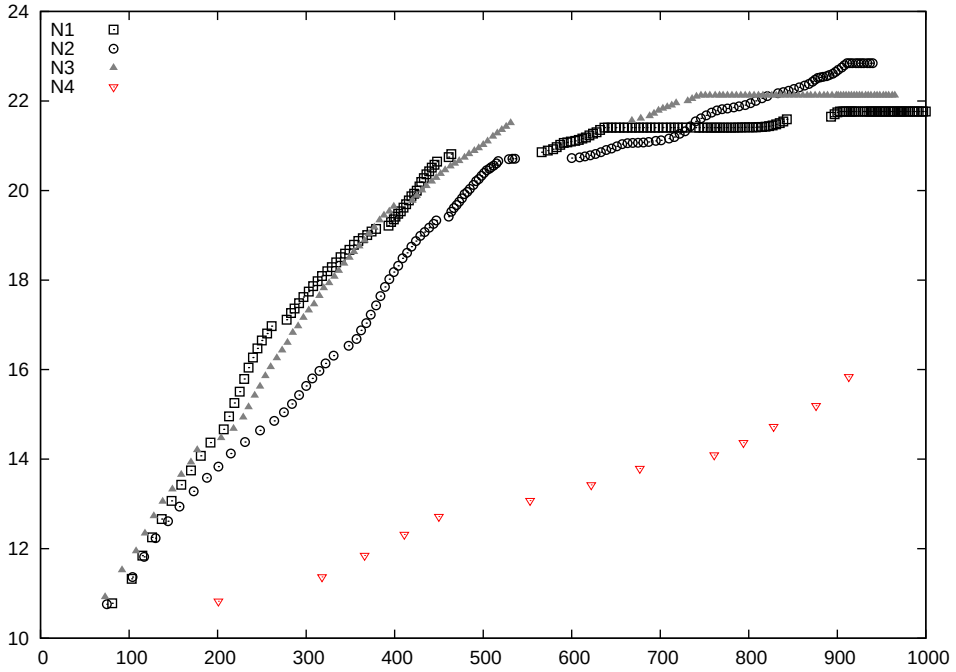


Figure 5: Average fitness in the first 1000 milliseconds of execution of the four nodes of the heterogeneous cluster with the same population sizes (HoSi/HeHa) for the MMDP problem.

of the migration is larger in the HeHa cluster because the network is having real conditions of traffic during the experiment. In the MMDP problem (Table 8) changing the population size does not affect the migration time, but it affects the rest of the algorithm's stages. However, with larger data communications (individuals of 5000 elements of the OneMax problem), the population size affects the migration time of all nodes. This might be due to the synchronization of migration buffers: if the slowest machine is sending/receiving, bottlenecks can be propagated (as it can seen in Figure 9).

Results also show how the stages of the algorithms depends on the node of execution. For example, recombination needs more time than mutation in both problems only in the node HeN4. The reason might be the creation of new objects (memory allocation), which in Java and in limited memory (and SWAP access) requires more time than iteration of elements previously created (for example, in the mutation). Adapting the population size makes the slower node of HeHa behaves in similar way than the other nodes (same time in each stage). Moreover, the size of the individuals affects some parts of the EA; for example, in the OneMax the mutation requires more time than the replacement. However, it must be taken into account that the duration of each part of the algorithm is not related with the time to attain the optimum, but to how the diversity and search guidance is maintained in the whole system.

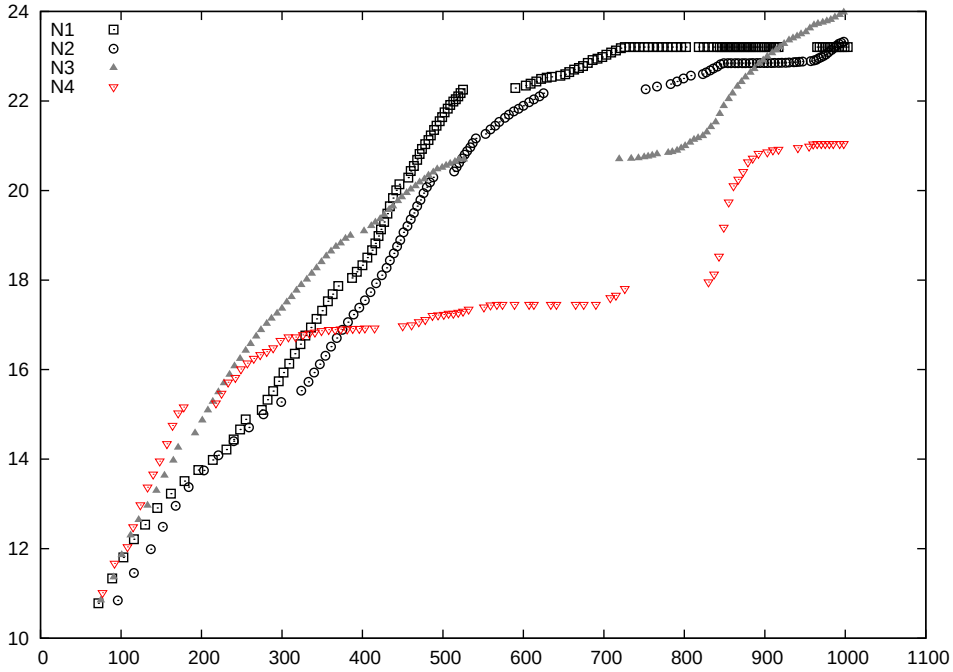


Figure 6: Average fitness in the first 1000 milliseconds of execution of the four nodes of the heterogeneous cluster with different population sizes (HeSi/HeHa) for the MMDP problem.

5. Conclusions

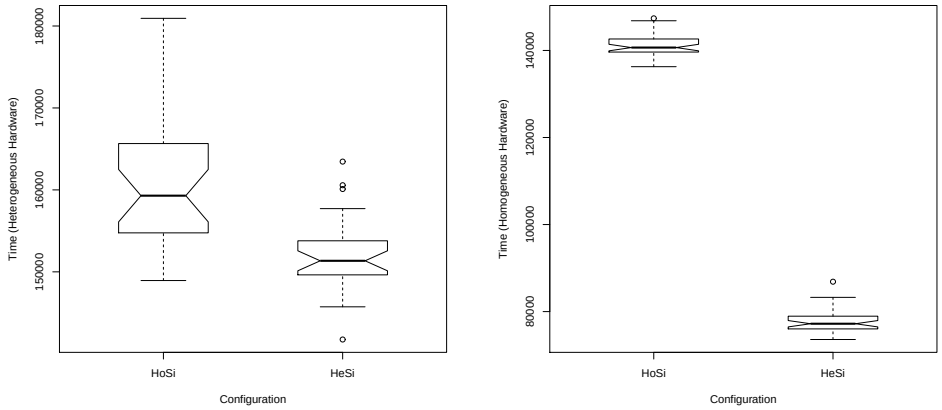
In this paper we describe a study on the adaptation of the population size of a distributed EA to the computational power of the different nodes of an heterogeneous cluster. The same parameter set has been also tested in a homogeneous cluster.

Results shows that adapting the population size to the computational power of each node in the heterogeneous cluster yields significantly better results in time than keeping the same parameter in all nodes. This advantage is due to the combination of the heterogeneous parameters with the heterogeneity of the machines. On the contrary, the same (heterogeneous) parameter set in all islands of the homogeneous cluster could not improve the results than the same parameter in all nodes.

Moreover, changing the population size affects to stages of the algorithm that are independent of the population size, such as the migration.

In this work, we calculate the computational power of each node proportionally to the average number of generations for the homogeneous parameter set. These results are a promising start for adapting EAs to the performance of each execution node, using more adequate benchmarks or in a dynamic way.

In the future it would be interesting to check the scalability of this approach, using more computational nodes and larger problem instances. In addition, other parameters such as migration rate or crossover probability could be adapted to the execution nodes. Different benchmarks will be also used to lead to automatic parameter adaptation in runtime (online), with different nodes



(a) Heterogeneous cluster

(b) Homogeneous cluster

Figure 7: Time to obtain the optimum in the OneMax problem (milliseconds).

entering or exiting in the topology, or adapting the parameters to the current load of the system.

Acknowledgements

This work has been supported in part by FPU research grant AP2009-2942 and projects EvOrq (TIC-3903), CANUBE (CEI2013-P-14) and ANYSELF (TIN2011-28627-C04-02).

[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] Mine Altunay, Paul Avery, Kent Blackburn, Brian Bockelman, Michael Ernst, Dan Fraser, Robert Quick, Robert Gardner, Sebastien Goasguen, Tanya Levshina, Miron Livny, John McGee, Doug Olson, Ruth Pordes, Maxim Potekhin, Abhishek Rana, Alain Roy, Chander Sehgal, Igor Sfiligoi, Frank Wuerthwein, and Open Sci Grid Executive Board. A Science Driven Production Cyberinfrastructure-the Open Science Grid. *Journal of GRID Computing*, 9(2, Sp. Iss. SI):201–218, JUN 2011.

[4] Lourdes Araujo and Juan Julián Merelo Guervós. Diversity through multiculturality: Assessing migrant choice policies in an island model. *IEEE Trans. Evolutionary Computation*, 15(4):456–469, 2011.

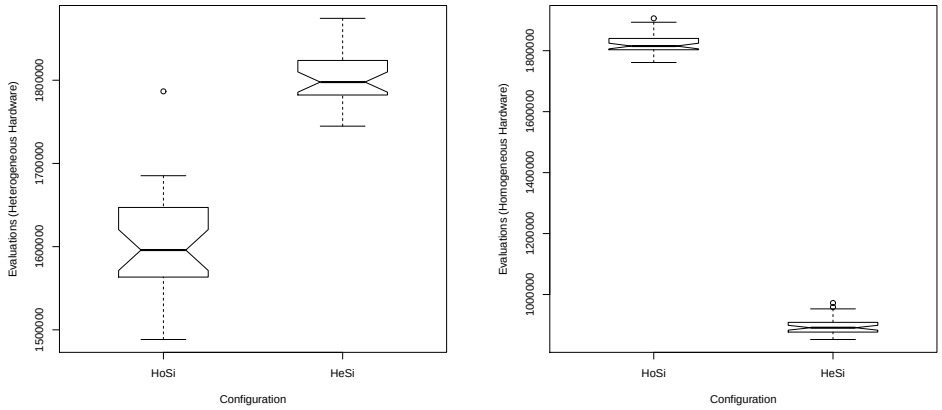
[5] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25:599–616, June 2009.

[6] Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.

[7] Julián Domínguez and Enrique Alba. HydroCM: A hybrid parallel search model for heterogeneous platforms. In El-Ghazali Talbi, editor, *Hybrid Metaheuristics*, volume 434 of *Studies in Computational Intelligence*, pages 219–235. Springer Berlin Heidelberg, 2013.

[8] A. E. Eiben and Selmar K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

[9] I Foster. Globus Toolkit version 4: Software for service-oriented systems. In Jin, H and Reed, D and Jiang, W, editor, *Network and Parallel Computing Proceedings*, volume 3779 of *Lecture Notes in Computer Science*, pages 2–13, 2005.



(a) Heterogeneous cluster

(b) Homogeneous cluster

Figure 8: Number of evaluations for OneMax problem.

[10] J.F. Garamendi and J.L. Bosque. Parallel implementation of evolutionary strategies on heterogeneous clusters with load balancing. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 8 pp., april 2006.

[11] P. García-Sánchez, J. González, P.A. Castillo, M.G. Arenas, and J.J. Merelo-Guervós. Service oriented evolutionary algorithms. *Soft Computing*, 17(6):1059–1075, 2013.

[12] 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.

[13] 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.

[14] 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.

[15] Sergio Nesmachnow, Hctor Cancela, and Enrique Alba. A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling. *Applied Soft Computing*, 12(2):626 – 639, 2012.

[16] Andres J. Ramirez, David B. Knoester, Betty H. C. Cheng, and Philip K. McKinley. Plato: a genetic algorithm approach to run-time reconfiguration in autonomic computing systems. *Cluster Computing*, 14(3):229–244, 2011.

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

[18] 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.

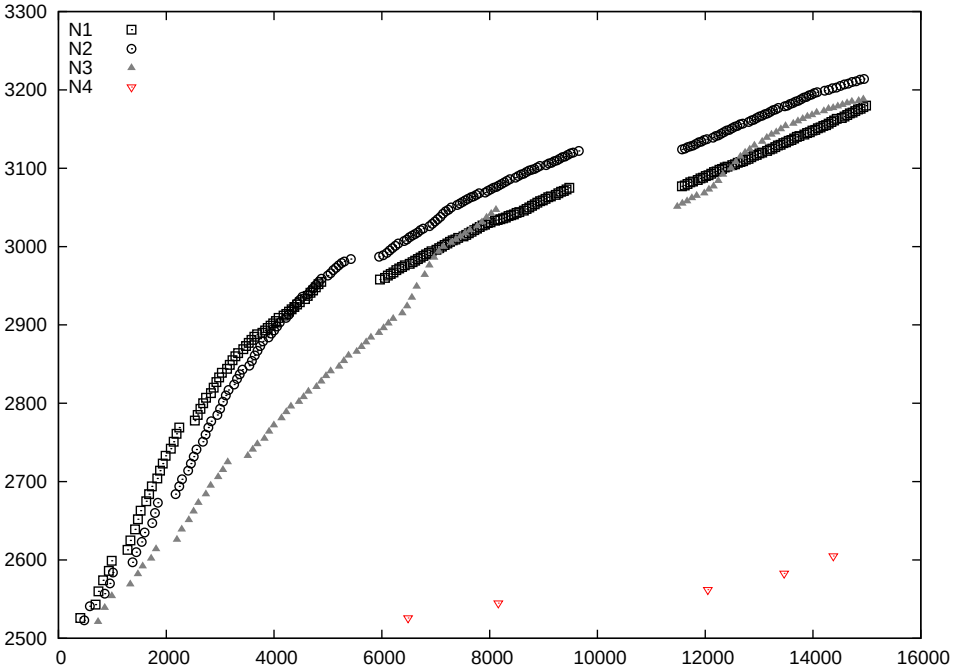


Figure 9: Average fitness in the first 15000 milliseconds of execution of the four nodes of the heterogeneous cluster with the same population sizes (HoSi/HeHa) for the OneMax problem.

Table 7: Statistical significance of the results.

Configuration	Normal	Test applied	P-value	Significant difference?
Time for MMDP				
HoSi/HeHa vs HeSi/HeHa	No	Wilcoxon	0.0009	Yes
HoSi/HoHa vs HeSi/HoHa	No	Wilcoxon	0.52	No
Evaluations for MMDP				
HoSi/HeHa vs HeSi/HeHa	No	Wilcoxon	0.002	Yes
HoSi/HoHa vs HeSi/HoHa	No	Wilcoxon	0.08	No
Time for OneMax				
HoSi/HeHa vs HeSi/HeHa	No	Wilcoxon	1×10^{-5}	Yes
HoSi/HoHa vs HeSi/HoHa	No	Wilcoxon	3×10^{-8}	Yes
Evaluations for OneMax				
HoSi/HeHa vs HeSi/HeHa	No	Wilcoxon	7×10^{-9}	Yes
HoSi/HoHa vs HeSi/HoHa	No	Wilcoxon	3×10^{-8}	Yes

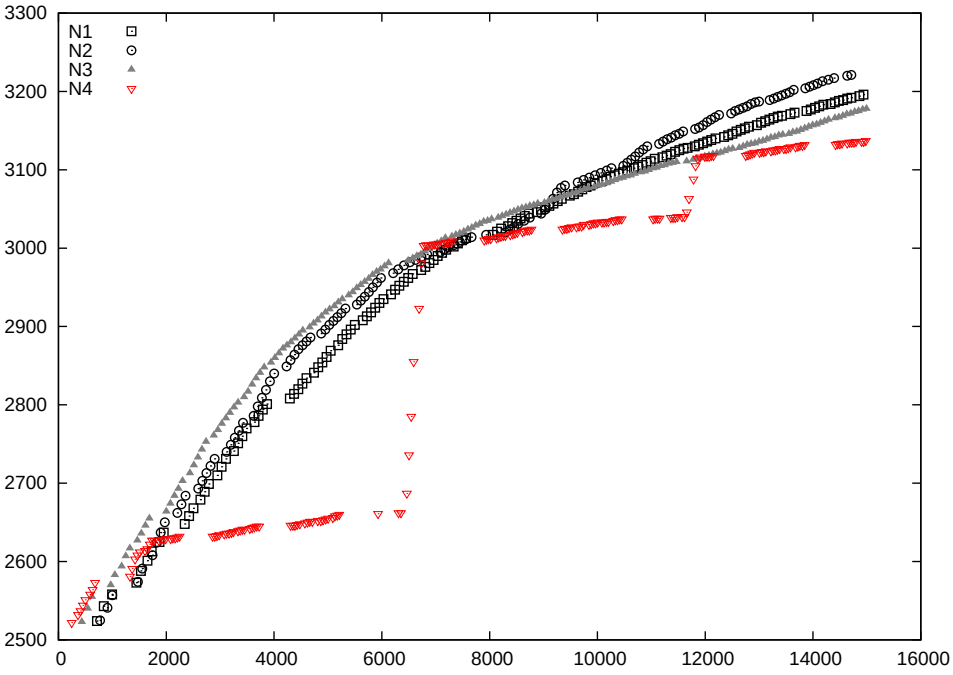


Figure 10: Average fitness in the first 15000 milliseconds of execution of the four nodes of the heterogeneous cluster with different population sizes (HeSi/HeHa) for the OneMax problem.

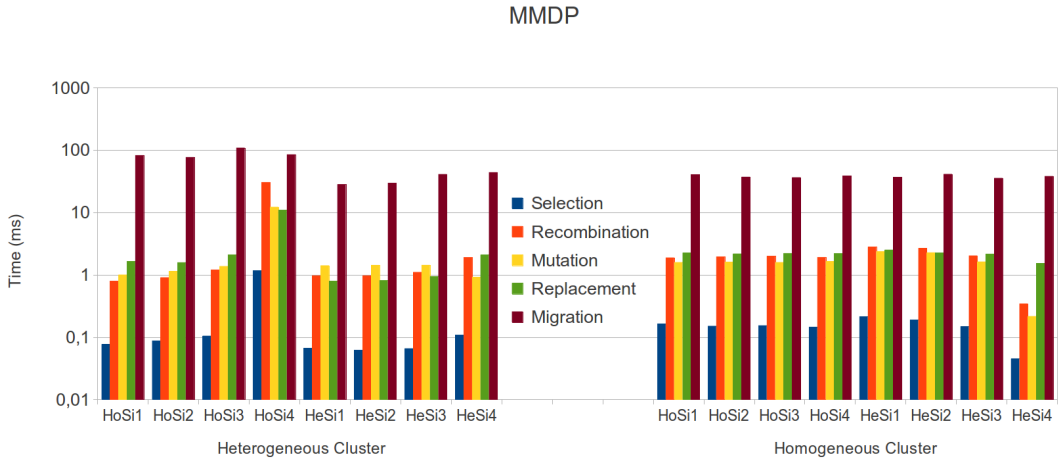


Figure 11: Average running time in each stage of the algorithm for the MMDP problem.

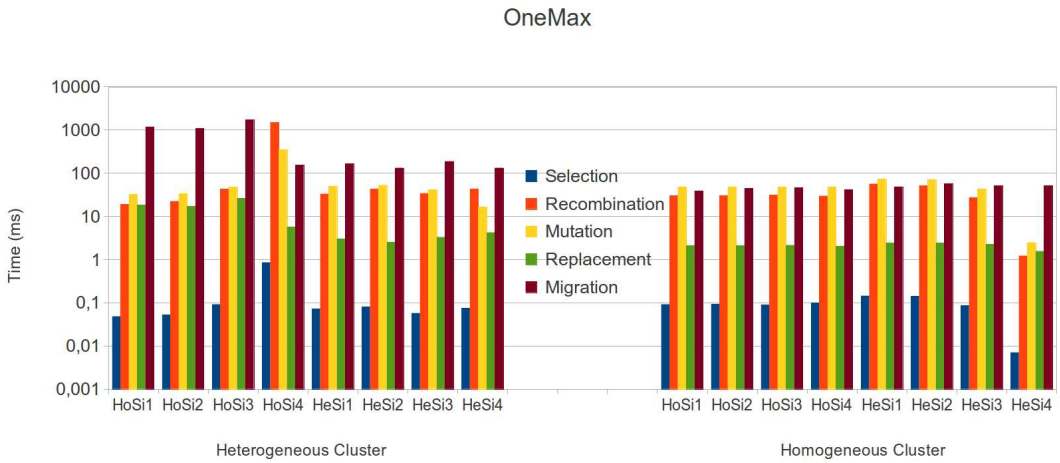


Figure 12: Average running time in each stage of the algorithm for the ONEMAX problem.

Table 8: Times of the stages of the algorithm for the MMDP problem (in ms).

Heterogeneous Cluster					
Node	Selection	Recombination	Mutation	Replacement	Migration
HoSi HeN1	0,077 ± 0,170	0,788 ± 0,779	1,004 ± 0,187	1,648 ± 20,185	82,458 ± 143,266
HoSi HeN2	0,088 ± 0,190	0,907 ± 0,932	1,145 ± 0,425	1,579 ± 17,907	76,725 ± 126,360
HoSi HeN3	0,105 ± 0,163	1,207 ± 0,927	1,374 ± 0,301	2,108 ± 21,848	108,605 ± 142,633
HoSi HeN4	1,165 ± 1,526	30,445± 59,553	12,221 ± 7,412	10,978 ± 57,135	84,936 ± 0,000
HeSi HeN1	0,067 ± 0,065	0,973 ± 0,403	1,411 ± 0,166	0,790 ± 6,266	28,081 ± 42,169
HeSi HeN2	0,062 ± 0,075	0,973 ± 0,470	1,433 ± 0,265	0,811 ± 7,056	29,667 ± 48,702
HeSi HeN3	0,066 ± 0,108	1,104 ± 0,346	1,435 ± 0,296	0,937 ± 7,072	40,964 ± 40,027
HeSi HeN4	0,109 ± 0,257	1,895 ± 5,611	0,913 ± 0,834	2,085 ± 5,626	43,880 ± 7,535
Homogeneous Cluster					
Node	Selection	Recombination	Mutation	Replacement	Migration
HoSi HoN1	0,163 ± 0,223	1,884 ± 2,386	1,591 ± 0,479	2,254 ± 5,513	40,256 ± 8,726
HoSi HoN2	0,151 ± 0,212	1,952 ± 2,876	1,597 ± 0,574	2,178 ± 4,922	37,110 ± 6,999
HoSi HoN3	0,154 ± 0,206	1,990 ± 3,010	1,591 ± 0,577	2,215 ± 4,743	36,413 ± 5,266
HoSi HoN4	0,146 ± 0,196	1,913 ± 2,697	1,651 ± 1,167	2,194 ± 5,124	38,429 ± 6,192
HeSi HoN1	0,214 ± 0,288	2,800 ± 3,793	2,359 ± 0,691	2,516 ± 4,706	36,972 ± 4,214
HeSi HoN2	0,190 ± 0,252	2,672 ± 3,902	2,277 ± 0,649	2,261 ± 4,546	41,171 ± 9,672
HeSi HoN3	0,148 ± 0,208	2,030 ± 3,161	1,623 ± 0,500	2,164 ± 4,512	35,551 ± 6,132
HeSi HoN4	0,045 ± 0,052	0,345 ± 1,121	0,217 ± 0,142	1,531 ± 4,856	38,106 ± 9,251

Table 9: Times of the stages of the algorithm for the OneMax problem (in ms).

Heterogeneous Cluster					
Node	Selection	Recombination	Mutation	Replacement	Migration
HoSi HeN1	0,048 ± 0,043	18,713 ± 13,454	31,984 ± 2,104	18,375 ± 197,676	1172,986 ± 1108,388
HoSi HeN2	0,052 ± 0,051	22,266 ± 22,716	33,553 ± 4,931	17,176 ± 180,580	1085,508 ± 995,382
HoSi HeN3	0,091 ± 1,005	42,634 ± 21,621	47,674 ± 0,546	26,094 ± 252,667	1708,402 ± 1207,925
HoSi HeN4	0,851 ± 0,435	1491,568 ± 1185,723	344,872± 6,634	5,655 ± 16,175	154,019 ±0,000
HeSi HeN1	0,072 ± 0,063	32,917 ± 26,792	49,103 ± 2,655	3,023 ± 27,647	163,479 ±157,172
HeSi HeN2	0,080 ± 0,092	43,001 ± 51,680	52,288 ± 13,210	2,527 ± 21,861	131,063 ±124,404
HeSi HeN3	0,057 ± 0,052	33,951 ± 15,063	41,375 ± 1,707	3,284 ± 30,170	186,467 ±163,906
HeSi HeN4	0,075 ± 0,107	42,443 ± 88,536	16,236 ± 12,028	4,194 ± 33,119	131,135 ±144,359
Homogeneous Cluster					
Node	Selection	Recombination	Mutation	Replacement	Migration
HoSi HoN1	0,091 ± 0,078	29,969 ± 21,459	47,445 ± 2,194	2,073 ± 6,970	38,782 ± 40,369
HoSi HoN2	0,093 ± 0,082	30,119 ± 22,029	47,247 ± 2,146	2,108 ± 7,440	44,303 ± 42,759
HoSi HoN3	0,089 ± 0,080	30,951 ± 21,904	47,103 ± 2,031	2,138 ± 8,006	46,107 ± 47,351
HoSi HoN4	0,098 ± 0,075	29,468 ± 20,876	47,086 ± 1,856	2,043 ± 7,491	41,458 ± 44,970
HeSi HoN1	0,144 ± 0,151	56,124 ± 48,229	72,811 ± 5,177	2,424 ± 9,056	48,165 ±57,798
HeSi HoN2	0,141 ± 0,152	51,226 ± 41,016	70,047 ± 4,152	2,427 ± 10,890	57,152 ±74,177
HeSi HoN3	0,086 ± 0,088	26,932 ± 20,460	42,963 ± 3,935	2,239 ± 8,658	51,014 ±49,648
HeSi HoN4	0,007 ± 0,008	1,215 ± 1,133	2,470 ± 0,098	1,553 ± 10,078	50,498 ± 63,983