Noname manuscript No.
(will be inserted by the editor)

Designing Robust Volunteer-based Evolutionary Algorithms

J.L.J. Laredo · P. Bouvry · D.L. González · F. Fernández de Vega · M.G. Arenas · J.J. Merelo · C. Fernandes

Pre-print version
Published version available at `http://dx.doi.org/10.1007/s10710-014-9213-5`

Abstract This paper tackles the design of scalable and fault-tolerant evolutionary algorithms computed on volunteer platforms. These platforms aggregate computational resources from contributors all around the world. Given that resources may join the system only for a limited period of time, the challenge of a volunteer-based evolutionary algorithm is to take advantage of a large amount of computational power that in turn is volatile. The paper analyzes first the speed of convergence of massively parallel evolutionary algorithms. Then, it provides some guidance about how to design efficient policies to overcome the algorithmic loss of quality when the system undergoes high rates of transient failures, i.e. computers fail only for a limited period of time and then become available again. In order to provide empirical evidence, experiments were conducted for two well-known problems which require large population sizes to be solved, the first based on a genetic algorithm and the second on genetic programming. Results show that, in general, evolutionary algorithms undergo a *graceful degradation* under the stress of losing computing nodes. Additionally, new available nodes can also contribute to improving the search process. Despite losing up to 90% of the initial computing resources, volunteer-based evolutionary algorithms can find the same solutions in a failure-prone as in a failure-free run.

Keywords Evolutionary computation · Distributed algorithms · Fault tolerance · Genetic programming · Genetic algorithms · Volunteer computing · Peer-to-Peer · Desktop Grid

J.L.J. Laredo and P. Bouvry
FSTC-CSC/SnT, University of Luxembourg, Luxembourg.
E-mail: {juan.jimenez,pascal.bouvry}@uni.lu

D.L. González
Citizen Cyberscience Centre, Geneva, Switzerland. E-mail: teleyinex@gmail.com

F. Fernández de Vega
University of Extremadura, Spain. E-mail: fcofdez@unex.es

M.G. Arenas and J.J. Merelo
ATC-ETSIIT, University of Granada, Spain. E-mail: {mgarenas,jmerelo}@geneura.ugr.es

C. Fernandes
Laseeb, University of Lisbon, Portugal. E-mail: cfernandes@laseeb.org

1 Introduction

Every single day millions of users are connected to the Internet [Stats(2013)]. In addition to the most traditional activities, such as web surfing or emailing, users also exhibit a more sophisticated behavior: they synchronize their devices in the cloud, play their favorites games on-line or talk using VoIP applications. More relevant is that they do it all in a natural and seamless way. Computation is turning out to be pervasive in our everyday life. If only a minor fraction of these users gets engaged to an Internet application, their aggregated CPU power may turn the whole system into a large-scale environment for distributed computation.

Relying on this idea, volunteer computation tries to aggregate computational power from volunteers from all over the Internet that are willing to donate their idle CPU cycles to different research projects. A volunteer system can be started from installing a client application in several computer devices at the edges of the network; for that aim, two of the most widespread technologies are desktop grids (DGs) [Anderson(2004)] and peer-to-peer systems (P2P) [Steinmetz and Wehrle(2005)]. Both approaches refer to distributed ad-hoc networks of heterogeneous single systems; however, DGs follow a centralized approach while P2P systems are decentralized. These distributed platforms have been postulated as an alternative to large-scale supercomputers for very specific problem domains, as in the well-known SETI@home project [Anderson et al(2002)] for the search of extraterrestrial intelligence patterns in radio signals from space.

Volunteer systems can also be applied to time-consuming parallel evolutionary algorithms (EAs). In fact, there is a growing interest of the scientific community in the paradigm and many optimization heuristics have been redesigned recently in order to take advantage of this aggregated source of computational power, e.g. [Desell et al(2008), Laredo et al(2010), Scriven et al(2008b), Biazzini and Montresor(2010), Bánhelyi et al(2009)]. Overall, the purpose is to increase the convergence speed via the massive CPU availability of volunteer systems. However, with large scale comes a higher likelihood that processors suffer a failure, or that volunteers disconnect/connect their resources at runtime, thus interrupting the raw execution of the algorithm or even crashing the whole system. Given that such dynamics (a.k.a. *host-churn* or simply *churn* [Stutzbach and Rejaie(2006)]) are inherent to systems of this kind, they must be considered when designing a volunteer application.

The aim of this paper is to study the scalability and fault tolerance of volunteer-based EAs and to provide some guidelines about how to design robust EAs in the context of different volunteer computing environments and churn scenarios: robustness understood as the property of an EA to guarantee consistency in results despite the volatility of resources.

To that end, two different volunteer platforms are studied: DG and P2P. To analyze each system under the same conditions, churn is simulated using real-world traces of host availability. Fault tolerance is then analyzed in terms of loss of quality in obtained solutions: we have firstly considered a worst-case scenario in which the systems degrade from an initial maximum of available hosts, i.e. the population size shrinks as the computers fail. Besides, in a more realistic test case, several policies have been designed to allow that new available computers rejoin the ongoing search. This results in the population changing size as computers arrive to or depart from the system; a merely accidental consequence which arises from the system dynamics. However, it directly links our approach to other variable population size studies, a technique that has been previously devised for optimizing EAs performance [Montero and Riff(2011)].

Finally, and seeking for the capacity of generalization for the obtained results, experiments were conducted for two of the most relevant paradigms in evolutionary computation (EC): genetic algorithms (GAs) and genetic programming (GP). For the GA case, we have considered a large instance of a *trap* function from the study of Thierens [Thierens(1999)] on the scalability of GAs. For the given instance, the population size requirements are high (i.e. 3000 individuals to find near-optimal solutions), and that allows the algorithm to be deployed in a large-scale infrastructure. For GP, we have considered the 11 multiplexer instance proposed by Koza [Koza(1992)] which also requires a large population size to guarantee a reliable convergence.

The rest of this paper is organized as follows: Section 2 surveys the state of the art in parallel evolutionary algorithms (PEAs) and focuses on volunteer-based EAs. Section 3 outlines the parallel approaches that will be used along this paper: a master-slave evolutionary algorithm for DG systems and a decentralized P2P evolutionary algorithm. Section 4 describes the methodology and settings followed in the experiments, including a description of the traces of churn and the tuning of the parameters. Section 5 presents the results of the experiments for three different test cases which respectively tackle the massive scalability of volunteer-based evolutionary algorithms, their resilience to system degradation and different strategies for recruiting new available computers into the system. Finally, some conclusions are drawn in Section 6 and some future lines of work proposed.

2 Related Work

Research in Parallel EAs (PEAs) is nowadays tightly coupled with advances in Computer Architecture. Standard models of parallelization, as classified by Alba and Tomassini [Alba and Tomassini(2002)], are being redesigned/hybridized to exploit the features of new parallel architectures such as graphic processing units [Pospichal et al(2010)], decentralized databases [Roy et al(2009)] or mobile ad-hoc devices [Karafotias et al(2011)].

Within the set of new parallel systems, Internet-based architectures represent one of the most attracting fields for researchers because, unlike in proprietary infrastructures, Internet provides a scale on-demand access to resources. Thus, new PEAs have been designed to take advantage of cloud computing systems [Arenas et al(2011), Derby et al(2013)], grid systems [Lim et al(2007)], or volunteer computing systems [Desell et al(2010), Biazzini and Montresor(2013), O'Reilly et al(2013)]. The latter has become popular since volunteer systems provide a unique infrastructure of massively large and potentially free resources for tackling challenging problems; the Milky-Way@HOME project [Desell et al(2008)], for example, is a time-consuming volunteer-based application that tries to find an optimal fit of three-dimensional models of the Milky Way galaxy to star data observed.

Despite having a common goal, the way of harnessing computing power in volunteer systems has diverged into two main technologies: DGs and P2P systems. In comparison, DG technology is more developed thanks to the efforts that David Anderson and his team at UC Berkeley invested in the development of the BOINC platform [Anderson(2004)] which has been used in most of the DG-based EAs designs [Desell et al(2010), Gonzalez et al(2009b), Varrette et al(2008)]. However, having a central master may become a problem as the master represents a single-point of failure and a bottleneck in data-intensive exchange applications. P2P systems, on the other hand, overcome this issue with a decentralized architecture in which ev-

ery computing node acts either as a client or as a server, thus limiting the impact a node can have on the system. Following such a decentralized philosophy, many approaches have been tackling the design of P2P meta-heuristics [Arenas et al(2002), Wickramasinghe et al(2007), Laredo et al(2010), Biazzini and Montresor(2010), Scriven et al(2009), Bánhelyi et al(2009), Biazzini and Montresor(2013)].

All in all, volunteer systems are highly dynamical environments in which resources may fail, join or depart from the network in an unpredictable way. This phenomenon, known as churn [Stutzbach and Rejaie(2006)], requires fault-tolerant mechanisms to circumvent the volatility of resources and to prescribe a defined behavior of an application once a failure occurs. A set of techniques such as checkpointing or matchmaking [Cahon et al(2005)] have been previously proposed to cope with failures in grid and cluster environments. However, such techniques would represent a bottleneck if applied to a large volunteer system because of their memory and synchronization overheads.

The research of fault-tolerant volunteer systems is more recent, and leverages in the concept of acquiring robustness as an emergent property of the algorithm design. The simplest approach is to analyze platforms which undergo a gradual loss of resources, *"the assumption being the decentralised structure of the algorithm itself would provide robustness"* [Scriven et al(2009)]. On the other hand, some other works (reviewed below) assume the replacement of nodes, an approach that consists in keeping a steady number of computing resources. Then, failures are simulated by restarting random nodes.

Scriven et al. [Scriven et al(2008a)] analyze the performance of a multi-objective particle swarm optimization algorithm in a failure-prone environment. The algorithm shows to be resilient under small failure rates but degrading in turn under high failure rates.

Bánhelyi et al. present in [Bánhelyi et al(2009)] both a P2P particle swarm optimization and a P2P branch-and-bound algorithm. Churn is then characterized by considering the replacement of one percent of nodes every twenty function evaluations. The restarting nodes are shown to help the algorithm in escaping from local optima and thus, to improve the global search.

Biazzini and Montresor [Biazzini and Montresor(2010)] propose a gossiping differential evolution algorithm where individuals are spread to different sub-populations using an epidemic migration. The proposal considers a similar characterization of churn as the previous work but replacing in turn five and ten percent of the nodes. Authors show how the algorithm is able to find good solutions by fine-tuning one of the parameters of the algorithms, namely gossip rate.

Scriven et al. in [Scriven et al(2009)] get back into their multi-objective particle swarm algorithm but this time under the perspective of restarting nodes instead of making them fail. The authors explore different methods for reinitializing the particles of a node. In the multi-objective context of the paper, the winning strategy increases the coverage of the pareto front by filling empty gaps with the new particles.

In previous works [Gonzalez and Fernández(2007), Hidalgo et al(2007)] we first analyzed the fault-tolerance nature of PEAs under several simplified assumptions. These initial results suggested that EAs exhibit a fault-tolerant behavior by default, encouraging to go a step further to study the fault-tolerance of large-scale distributed EAs. In [Laredo et al(2008b)], we firstly focused on P2P systems to analyze fault tolerance in distributed EAs. Then, in [Gonzalez et al(2009a)] and [Gonzalez et al(2010)] we focused on DGs. The results show again that EAs can cope with failures without using any fault-tolerance mechanism, concluding that EAs are fault tolerant by nature since

they implement by default the fault-tolerance mechanism called *graceful degradation* [Ghosh(2006)].

From the above studies, there are two main outcomes that can be learned: the first is the inherent robustness of PEAs to failures; the second is the possibility of applying breeding strategies when nodes are restarted. However, it has to be noted that none of the previous works tackles both the departing and arrival of nodes at the same time, i.e. the real dynamics of a volunteer system. In addition, every single study establishes its own characterization of churn, which prevents to conclude that volunteer-based evolutionary algorithms are viable in general; especially, when a maximum failure rate of 20% is considered in most of the cases (e.g. [Scriven et al(2008a), Biazzini and Montresor(2010), Bánhelyi et al(2009), Scriven et al(2009)]).

In order to provide a realistic assessment on previous issues, we have conducted an *in-silico* experimentation which reproduces real-world traces of volunteer systems; e.g. in the worst-case scenario the system loses up to 90% of the initial resources. Furthermore, not only the resilience to node failures is analyzed but also the design of simple strategies that can be applied to new available incoming nodes. Despite their simplicity, such strategies are shown to succeed in circumventing churn dynamics: volunteer-based EAs are able to yield the same quality of solutions under churn as in a run without failures.

3 Description of the Models

This section describes the two EC models that are used in this paper for conducting experiments. The first one, in Section 3.1, is based on DGs and follows a canonical master-slave model. The second one, in Section 3.2, is a decentralized approach based on P2P technology.

We assume that both models implement a synchronous scheme, which helps to establish upper bounds on the degradation of the algorithms. The asynchronous approach, as shown by Scriven et al. [Scriven et al(2008a)], is intrinsically more robust to failures and therefore exhibits a more moderated degradation.

3.1 The centralized approach (DGs)

Desktop grid systems feature a master-slave paradigm, which naturally fits with the parallel evolutionary algorithm model: *parallelism at individual level* [Gagné and Parizeau(2003)]. In this paper, we follow the approach proposed in [Gonzalez et al(2010)] in which there is only a panmictic population, and the evaluation of the individuals' fitness is distributed, and thus shared, among several processors or nodes from the network. The rest of the operations (selection, reproduction, crossover and mutation) of the algorithm is carried out in the master node or server (see Fig. 1).

In general, the master server will host the population, send a fixed number of individuals to all the accessible workers and wait until all of them have been evaluated to breed the next generation. The length of the population will be equally divided among the available worker nodes, sending packages of individuals to get evaluated in each node.

Given the centralized nature of the approach and that the master represents a single-point of failure that will end up with the system crashing in case of failure, we

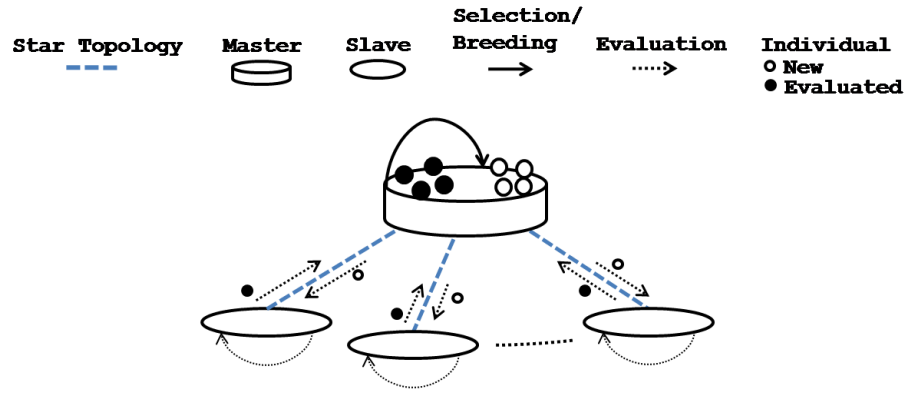


Fig. 1 Outline of the master-slave approach.

assume that computer failures are only possible in worker nodes. To avoid such problems in reality, the master is typically managed by institutions which adopt technical solutions to offer 24/7 availability [Anderson(2004)].

3.2 The decentralized approach (P2P)

Peer-to-peer computing is the other main approach to volunteer computing [Zhao et al(2009)]. In order to conduct experiments in a P2P-based volunteer system, this paper follows the evolvable agent model (EvAg), a decentralized approach proposed by the authors in [Laredo et al(2008a)] and extended in [Laredo et al(2010)] for massively large systems.

The EvAg model is a fine-grained spatially structured EA where *fine-grained* means that every agent schedules the evolution process of a single individual, and *spatially structured* means that the population structure is defined as a graph. Therefore, every agent can be represented as a vertex with edges to other neighbor agents. To make the algorithm inherently suited for parallel execution in P2P systems, the EvAg model defines the population structure as a P2P overlay network. To that end, the approach uses the newscast protocol, a gossiping and decentralized P2P protocol defined by Jelasity and van Steen in [Jelasity and van Steen(2002)].

Newscast follows a probabilistic scheme for exchanging routing information peer-to-peer. The loosely-coupled run of the protocol establishes a self-organized small-world connectivity between members, which grants a scalable way for disseminating information and enables the system for distributed computing. As depicted in Fig. 2, newscast determines that way the neighborhood of every EvAg and thus, constraints the mating scope of an agent among its neighbors.

Voulgaris et al. [Voulgaris et al(2004)] show that newscast is a robust protocol so that churn does not disrupt the graph structure. In order to illustrate such a resilience, Fig. 3 reproduces some of the results of the newscast seminal paper [Jelasity and van Steen(2002)] showing that the protocol is robust to nodes removal. It can be seen how newscast inherits the robust behavior of random graphs, especially when the node degree c is large (i.e. $c = 40$). On the one hand, the graph remains connected until a large percentage of nodes is removed; for $c = 40$, 90% of the nodes have to be removed to get a partition of the graph. On the other hand, most of the nodes remain in the larger cluster once the

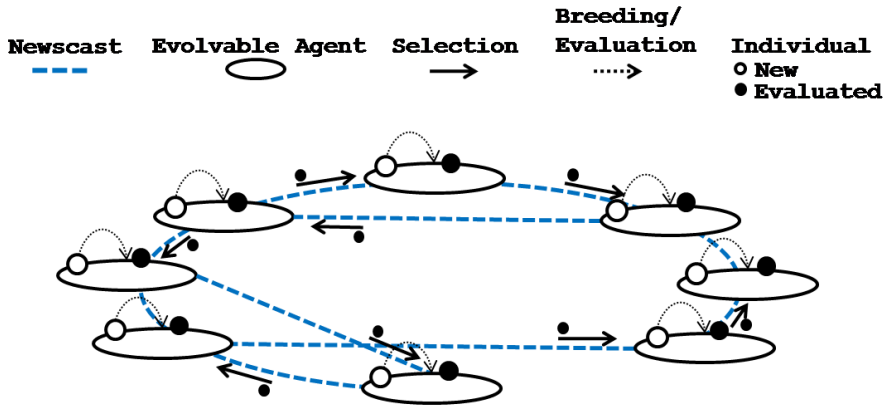


Fig. 2 Outline of the EvAg model using a newscast topology.

partition takes place. Therefore, in order to secure a resilient graph, all experiments in this paper are based on $c = 40$.

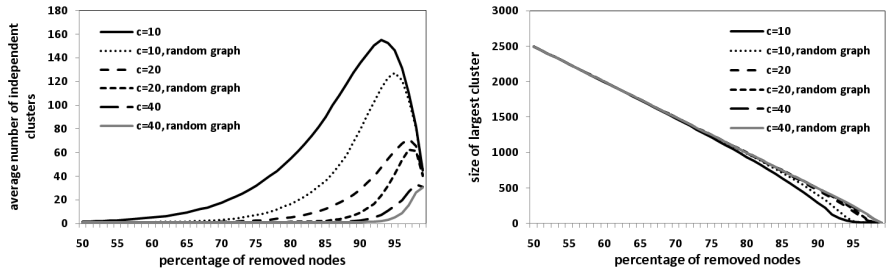


Fig. 3 Partitioning of the graph as a function of the percentage of removed nodes in a random graph and the respective newscast graph. c is a preset parameter which stands for the node degree. Results are averaged from 50 independent runs and a network size of 5000 nodes.

4 Experimental Setup

All experiments in this paper are conducted in the EvoGen simulator[1]. Simulations are a common approach for characterizing large-scale systems (e.g. in [Wickramasinghe et al(2007), Biazzini and Montresor(2010), Scriven et al(2009), Bánhelyi et al(2009), Gonzalez et al(2009b), Varrette et al(2008)]), especially when the aim is to perform a statistically significant number of experiments in a wide range of scenarios: each experiment in this paper has been evaluated over 100 independent runs. Furthermore, our experiments are repeatable via "replying" host availability trace data collected from real-world volunteer

[1] Source code for the experiments is available at `https://forja.rediris.es/svn/geneura/peerevogen`, published under GPL v3 public license.

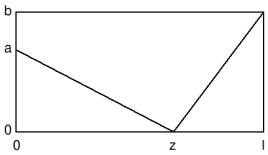


Fig. 4 Generalized *l-trap* function.

platforms [Kondo et al(2007)], so that a fair comparison between different simulated applications is possible.

Experiments are performed for two well-known GA and GP problems seeking for the capacity of generalization in obtained results. The GP problem is the 11-multiplexer (11M) problem which consists in learning the boolean 11M function described by Koza in [Koza(1992)]. The function involves the decoding of a 3-bit binary address (a_0, a_1, a_2) into the corresponding data register $(d_0, d_1, d_2, d_3, d_4, d_5, d_6, d_7)$, e.g. $(0, 0, 0)$ addresses the register d_0, $(0, 0, 1)$ does with d_1 and so on. Given that the 11M is one of all possible boolean functions of 11 arguments (i.e. $3_a + 2_d^3$), the search space for the problem is of size 2^{2048}. In order to apply GP to the problem, Koza defines the set of terminals as $T = \{A0, A1, A2, D0, \ldots, D7\}$ and the set of functions as $F = \{AND, OR, NOT, IF\}$ which satisfies the closure property.

For the GA problem we used an instance of trap functions [Ackley(1987)]. Traps are piecewise-linear functions defined on unitation (the number of ones in a binary string) with two distinct regions in the search space, one leading to a global optimum and the other leading to the local optimum (see Fig. 4). In general, a trap function of l bits is defined by the following equation:

$$trap(u(\overrightarrow{x})) = \begin{cases} \frac{a}{z}(z - u(\overrightarrow{x})), & \text{if } u(\overrightarrow{x}) \in [0, z] \\ \frac{b}{l-z}(u(\overrightarrow{x}) - z), & \text{if } u(\overrightarrow{x}) \in]z, l] \end{cases} \quad (1)$$

where $u(\overrightarrow{x})$ is the unitation function returning the number of one values in bit string $\overrightarrow{x}$, a is the local optimum, b is the global optimum, l is the problem size and z is a slope-change location separating the attraction basin of the two optima. In this paper, l was set to 3 bits (i.e. 3-trap) which results in a quasi-deceptive problem for the following parameter values: $a = l - 1$, $b = l$, $z = l - 1$. The instance was then obtained by juxtaposing $m = 10$ blocks of 3 bits. These settings result in a problem of length $L = 30$ bits where the fitness can be computed by summing up the partial fitness of each sub-function m, and where the global optimum corresponds to $L = 30$.

In order to find optimal solutions, both problems require large population sizes which allows problems to be deployed in massively parallel volunteer systems via the proposed fine-grained parallelizations. Table 1 shows the parameter setup for the experiments. GP parametrization follows the settings proposed by Koza in [Koza(1992)] while GA parameters rely on the study of Thierens in [Thierens(1999)] on the scalability of trap functions. This includes optimal population sizes for both problems: choosing smaller sizes leads to premature convergence, while larger sizes will slow down the convergence speed. The optimal population size guarantees a predetermined success rate at the fastest convergence rate.

The general setting to every experiment is that at the onset of each generation every node has an equal number of individuals to evaluate. We call this number I and it assumes a system composed of homogeneous nodes for the sake of simplicity. In actual

Paradigm	GP	GA
Problem instance	11 bits multiplexer	3-Trap L=30
Optimum	0	30
Problem type	minimization	maximization
Population Size	4000	3000
Generations	50	30
Elitism	Yes	Yes
Crossover Operator	Koza's Bias Tree Crossover	Uniform Crossover
Crossover Probability	0.9	1.0
Mutation Operator	Koza's Subtree mut. (only applied in local search)	bit-flip
Mutation probability	-	$\frac{1}{L}$
Internal node prob. (subtree mutation)	0.9	-
Selection	Tournament (7)	Tournament (2)
Max Depth of Tree	17	-
DG Algorithm	Master-Slave GP	Master-Slave GA
P2P Algorithm	EvAg GP	EvAg GA
Node degree c (P2P)	40	40

Table 1 Settings of the algorithms

applications, it is a standard procedure to approach a homogeneous system behavior by dynamically load-balancing virtual nodes (a.k.a. *virtual servers*): *"To handle heterogeneity, each node picks a number of virtual servers proportional to its capacity"* [Godfrey and Stoica(2005)].

With P individuals to be evaluated at the first generation and N nodes, each node evaluates $I = \frac{P}{N}$ individuals. When a node fails, I individuals are lost and the population size shrinks. Given that these individuals are discarded for the next generation, the remaining nodes will continue evaluating I individuals each, regardless of the number of failures or newly available hosts.

The simulation of host availability is performed based on three real-world traces of host availability that were measured and reported in [Kondo et al(2007)]: *ucb*, *entrfin*, and *xwtr*. These traces are time-stamped observations of the host availability in three volunteer systems. In order to map such traces into the simulations, a simulator cycle is assumed to correspond to a 10 seconds interval of the data observed. The *ucb* trace was collected for 85 hosts in a graduate student lab in the EE/CS Department at UC Berkeley for about 1.5 months. The *entrfin* trace was collected for 275 hosts at the San Diego Supercomputer Center for about 1 month. The *xwtr* trace was collected for 100 hosts at the Université Paris-Sud for about 1 month. See [Kondo et al(2007)] for full details on the measurement methods and the traces, and Table 2 for a summary of the main features.

Table 2 Features of Desktop Grid Traces

Trace	Hosts	Time in months	Place
entrfin	275	1	SD Supercomputer Center
ucb	85	1.5	UC Berkeley
xwtr	100	1	Université Paris-Sud

Independently of the approach in use, if there is host churn then the population size will increase or decrease at run-time according to the availability of hosts. We impose the restriction that the overall population never overcomes the specified initial population size. This sets a failure-free run as an upper bound for computational resources. However, it may also leave some nodes idle in case a large number of nodes becomes available. In that case, it would be interesting to re-adjust the number I of individuals in every node in order to use all the available computing power. We leave such a load-balancing study outside the scope of this paper and maintain I constant.

5 Analysis of Results

This section studies the previously explained parallel approaches in a set of three different test cases. The aim is to characterize volunteer evolutionary computation in terms of time profits, solution qualities and fault-tolerance to computer failures.

5.1 Improving convergence speed

As stated in the introduction, the main motivation behind any massively parallel optimization model is to improve convergence speed in time-consuming problems, while preserving the quality of final solutions. Therefore, our first aim is to provide some insight into the computational and algorithmic performance of our parallel approaches. To that end, this first test case assumes an idealistic failure-free scenario where the available computing power is kept steady throughout the execution and where the number of resources is unlimited. It means that every individual is evaluated in a single node that lasts without failures until the end of the experiment.

A generational EA with 1-elitism is used as a baseline for comparison. The generational EA is run sequentially instead of in parallel but otherwise is algorithmically equal to the DG model (which follows a master-slave approach with the master hosting the evolutionary loop). This allows not only the comparison of the best solutions found at the maximum number of generations, but also a forecast on the time that every approach would take. Obviously, results are for guidance only since they assume negligible costs in communications and an ideal failure-free environment.

Fig. 5 shows the best fitness convergence curves for the sequential and parallel approaches. It is depicted in terms of simulator cycles and represented in a semi-log scale to appreciate differences in time of convergence. Every simulator cycle takes the time of an individual evaluation. Given that we have assumed no cost in communications, the algorithm speeds up proportionally to the population size (i.e. $speedup = 4000$ in the 11M problem and $speedup = 3000$ in 3-trap). In order to reproduce such speedups in a real setting, we would require more demanding problems than those addressed in this paper. However, as we demonstrated in [Laredo et al(2012)], quasi-linear speedups are feasible in large-scale systems when tackling time-consuming fitness evaluation functions, i.e. an increasing ratio between computation and communication favors scalability. In that sense, this first set of experiments has the purpose of representing an upper bound in the computational performance when parallelizing evolutionary algorithms in massively large platforms as volunteer-based systems are.

With respect to the fitness convergence, the P2P approach shows a better progress in fitness and is able to outperform the best solution found by the DG system at

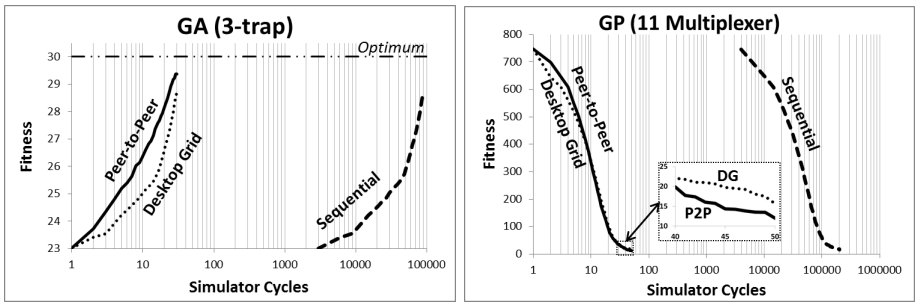


Fig. 5 Best fitness convergence of the peer-to-peer and desktop grid approaches with respect to the sequential approach when tackling the 3-trap (*left*) and the 11 bits multiplexer (*right*) problems. Representation is in *semi-log* scale and depicts ideal differences in simulator cycles between tackling the problems sequentially and in parallel. Results are averaged from 100 independent runs.

Table 3 Wilcoxon test comparing the best fitness distributions of equally parametrized DG and P2P approaches in the 3-trap (*maximization*) and 11M (*minimization*) problems. Results are obtained on 100 independent runs. We have considered that $p - values < 0.05$ refute the null hypothesis on the equivalence of the distributions.

Problem	Algorithm	Fitness	Wilcoxon test	Significantly different?
3-trap	DG	28.5±0.94	W = 2434	
	P2P	**29.38±0.66**	p-value = 4e-11	yes
11M	DG	22.18±45.5	W = 5677.5	
	P2P	**11.92±32.78**	p-value = 0.02	yes

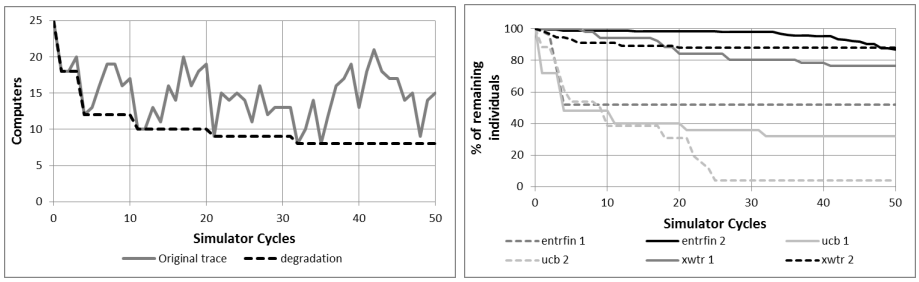
the maximum number of generations. Given that the algorithms have been equally parametrized, the most remarkable difference relies on the different population structures and breeding schemes. While the DG approach is panmictic, the P2P approach adopts the shape of a complex network and implements a decentralized breeding scheme.

Table 3 provides the Wilcoxon analysis of the best fitness showing significant differences in the 3-trap and 11M problem with the P2P approach outperforming the DG approach. Therefore, it can be concluded that the P2P approach is at least algorithmically competitive against DG which follows a canonical evolutionary scheme for breeding.

5.2 System degradation

In this section, preliminary experiments are conducted under the perspective of a failure-prone scenario in which the system loses resources. To that aim, simulations reproduce the traces of host availability described in section 4, namely *entrfin*, *ucb* and *xwtr*. Besides, we pose a stringent assumption in the simulations: hosts that become unavailable never become available again, i.e. the system degrades. Two random non-

overlapping segments from every trace were selected for simulations so that we ensure a good sample set to characterize host availability in real-world systems.[2]



(a) (b)

Fig. 6 Host availability for one sample of the *ucb* trace (*a*). Degradation of the population size at run-time for the six sampled traces (*b*).

Fig. 6(a) shows an example of available computers from the *ucb* trace and how the degradation takes place. The figure depicts the typical churn phenomenon, with available hosts becoming unavailable and later becoming available again. Given the condition of no host return, the curve "degradation" represents the availability of hosts along the algorithm run.

The six samples used in this study are depicted in Fig. 6(b). They represent the degradation which the evolutionary algorithm will suffer at run-time for each of the selected segments. Depending on the case, the population size will degrade following different failure rates going from the smooth degradation of the *entrfin 2* sample to the steep one in *ucb 2*. This translates into each trace establishing a different number of fitness evaluations at the end of a run. Table 4 compiles these results and also computes the volatility of resources in every trace as:

$$1 - \frac{\sum_{i=0}^{t_{max}} \frac{c_i}{c_{max}}}{t_{max}} \tag{2}$$

where t_{max} is the length of the trace in simulator cycles; c_i is the number of available computers at time $i \in [0, \ldots, t_{max}]$; and $c_{max} = max(c_i)$ the peak number of available computers in the trace.

Under these conditions, experiments were conducted in order to assess the resilience of volunteer evolutionary computation to failures. To that aim, results were compared to those obtained in the failure-free scenario of the previous section. Table 5 provides a statistical study of the best fitness distributions comparing the error-free fitness (*Eff*) with the failure-prone scenarios for the different approaches.

At a first glance, the statistical analysis of the table shows that every trace has a similar impact on the algorithmic performance independently of the parallel approach in use (*P2P* or *DG*) or the problem itself (*3-trap* or *11M*). The optimization processes suffer that way a graceful degradation in three of the scenarios where results are statistically equivalent to the failure-free runs. It is possible to tolerate a gradual loss of

[2] These samples are available together with the source code for the experiments at `https://forja.rediris.es/svn/geneura/peerevogen`.

Trace	FEs		Volatility	
failure-free	200000	90000	-	-
entrfin 1	108000	51000	0.46	0.43
entrfin 2	192000	87000	0.04	0.03
ucb 1	80000	39000	0.60	0.56
ucb 2	48000	33000	0.76	0.63
xwtr 1	172000	81000	0.14	0.10
xwtr 2	176000	81000	0.12	0.10
	GP	**GA**	**GP**	**GA**

Table 4 Number of fitness evaluations (FEs) and volatility for the different traces and approaches (GA and GP) in the degradation scenario.

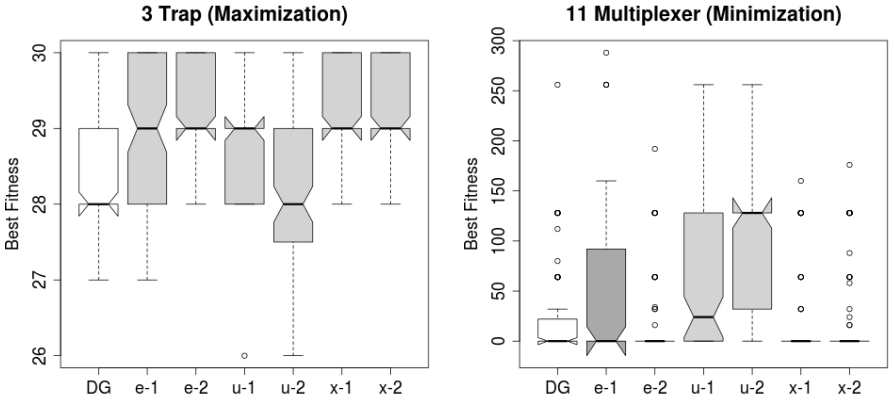
Traces	3-trap (Maximization)		11 Multiplexer (Minimization)	
	DG Eff = 28.5±0.94	P2P Eff = 29.38±0.66	DG Eff = 22.18±45.5	P2P Eff = 11.92±32.78
entrfin 1	F = 28.29±1.14 W = 5501 p-value = 0.19 ∼	F = 29.01±0.81 W = 6243 p-value = 0.001 −	F = 48.08±63 W = 4034 p-value = 0.0059 −	F = 47.84±68 W = 3539.5 p-value = 7e-06 −
entrfin 2	F = 28.63±1.01 W = 4588 p-value = 0.29 ∼	F = 29.31±0.66 W = 5296 p-value = 0.43 ∼	F = 20.56±43 W = 5265 p-value = 0.39 ∼	F = 15.86±39 W = 4836 p-value = 0.53 ∼
ucb 1	F = 27.84±1.1 W = 6677 p-value = 1e-05 −	F = 28.93±0.77 W = 6598 p-value = 2e-05 −	F = 63.62±66 W = 3225 p-value = 1e-06 −	F = 55.08±65 W = 3041 p-value = 1e-08 −
ucb 2	F = 26.86±1.05 W = 8616 p-value = 1e-19 −	F = 28.1±0.92 W = 8492 p-value = 6e-19 −	F = 103.12±65 W = 1531 p-value = 6e-19 −	F = 96.28±67 W = 1659 p-value = 2e-19 −
xwtr 1	F = 28.4±0.97 W = 5156 p-value = 0.68 ∼	F = 29.28±0.7 W = 5355 p-value = 0.34 ∼	F = 20.9±43 W = 5300 p-value = 0.33 ∼	F = 21.12±45 W = 4657 p-value = 0.21 ∼
xwtr 2	F = 28.43±1 W = 5135 p-value = 0.73 ∼	F = 29.2±0.7 W = 5374 p-value = 0.32 ∼	F = 17.6±40 W = 5467 p-value = 0.13 ∼	F = 19.46±43 W = 4405 p-value = 0.24 ∼

Table 5 Best fitness comparison between error-free (Eff) and error-prone fitness (F) cases using Wilcoxon test – "∼" means fitness quality is comparable to the error-free case while "−" stands for a degradation in the fitness quality.

up to 24% of the individuals (i.e. 14% of the computational effort) without sacrificing solution quality and more important without using any fault-tolerance mechanism. However, if the loss of computational power is too high, that is above 43%, the solution quality is significantly diminished. From all the sampled traces, *ucb 2* has obtained the worst fitness. The reason is that for such a trace the evolutionary algorithm loses up to 95.83% of the population. Consequently it is very difficult for the algorithm to obtain a solution with a similar quality as the error-free scenario.

On the other hand, an analysis based on the data provides better insights into the differences between the approaches and how degradation affects the performance. In general, the quality of solutions degrades smoother than system resources do, e.g.

for the *entrfin1* trace, the quality of solutions is only diminished by a $\sim 2\%$ in spite of the system losing up to 45% of the initial resources. Given that such a smooth degradation is consistent independently of the problem, trace or algorithmic approach, we can conclude that volunteer evolutionary algorithms degrade gracefully. However, it has to be noted that the P2P algorithm performs better than the centralized DG approach. For instance, if we consider the results of the P2P approach for the 3-trap instance in the *ucb 1* trace, it obtains a mean best fitness of 28.93. The revealing fact is that such a value still outperforms the mean best fitness of the error-free run in the DG approach (i.e. $Eff = 28.5$). Although the *ucb 1* trace degrades the system up to a 64% of the initial resources, the optimization process in the P2P approach is able to find near-optimal solutions still outperforming the failure-free run of the DG approach.



(a) *Compilation of results for 3-trap* (b) *Compilation of results for 11 Multiplexer*

Fig. 7 Error-free fitness distributions for the desktop grid approach (*white*) and failure-prone scenarios for the peer-to-peer approach (*gray*). "e-1" and "e-2" stand for entrfin traces, "u-1" and "u-2" for ucb and "x-1" and "x-2" for xwtr.

To compare performances of both approaches in detail, Fig. 7(a) and 7(b) show distributions on the best fitness for the failure-free run of the DG approach and the failure-prone runs of the P2P approach. Despite failures, the P2P algorithm outperforms the DG approach in 3-trap for almost every case except for the *ucb 2* trace where the system ends with a 4% of the initial resources. In the 11 Multiplexer problem, the P2P algorithm also outperforms the failure-free run of the DG algorithm in three out of six traces.

5.3 Testing the effect of a variable population size

Studying the inherent fault-tolerance of evolutionary algorithms, i.e. the way they degrade gracefully, provides some insights into the robustness of volunteer evolutionary computation; however, real-world volunteer systems implement nodes with rejoining abilities so that resources can become available again and can be reused by the application. This phenomenon is called *churn*. To take full advantage of such dynamics,

the system has to provide active fault-tolerant policies discerning what to do with new available nodes arriving.

Unlike in the degradation scenario, in which the population size becomes progressively smaller during the run, a churn-aware policy allows a variable population size scheme by assigning new individuals to new available nodes. The important question here is: what is assigned to newly available nodes? To gain insights into this question, this test case analyzes two different policies, one based on *breeding* techniques and the other one based on *local search*. Both policies apply the same working principle: once a new node becomes available in the system, I new individuals are created and assigned to it so that the global population size increases. Without any loss of generality, our aim is to show that, despite their simplicity, both policies are able to increase the robustness of a volunteer-based EA. Nevertheless, other new policies could also be developed by following the same structure, e.g. initializing randomly the new I individuals.

The difference between the proposed policies relies on the method they use to generate the I individuals.

- The *breeding* policy does not alter the reproduction scheme of the ongoing search; it simply breeds I more individuals whenever a new node arrives.
- On the other hand, the *local search* policy transforms our approach into a hybrid algorithm, where the population-based global search of the evolutionary algorithm is coupled with a local search phase at the arrival of nodes. The process is equivalent to the *breeding* policy except for the treatment of the I selected individuals. Instead of breeding them with crossover, only mutation is considered so that each of the new I individuals performs a local search in the neighborhood of the selected solution, the step size of the local search being determined by the amount of change introduced by mutation. We have considered *bit-flip* mutation for the GA approach and *Koza's subtree* mutation for GP (full details in Table 1).

Note that the implementation of both policies will differ between DG and P2P approaches:

- In the DG approach it is the master which has to be notified on the arrival of a node. Once the master is notified, it generates I new individuals which are assigned to the new node for evaluation.
- In the P2P approach, every new available node clones I new individuals after starting from a neighbor node, that is, the process is decentralized. Since nodes are disconnected from the network when they are switched off, another question is how they can rejoin the system. To that aim, the *newscast* protocol only requires the new node to be aware of a single connected node. Therefore, we assume that a node will always know an entry point to the connected network[3].

Table 6 shows the yielded results for the *breeding* and *local search* policies using DG and P2P approaches. The statistical analysis on every square represents the comparison between the given fault-tolerant policy and the respective distribution if only degradation is assumed. That way, "+" symbols denote that the given policy outperforms the results with respect to the system degrading (i.e. the policy reports benefits to the algorithm performance), "$\sim$" means that the algorithm behaves the same way

[3] This issue has been solved by the P2P computing community with the use of *hubs* or *superpeers*, a set of public and reliable nodes keeping a partial knowledge of the connected network. See [Steinmetz and Wehrle(2005)] for more details.

Breed

Traces	3-trap (Maximization)		11 Multiplexer (Minimization)	
	DG	P2P	DG	P2P
entr. 1	F = 28.67±1.02 W = 4038 p-value = 0.01 +	F = 29.46±0.6 W = 3479 p-value = 5e-05 +	F = 20.96±44 W = 6189 p-value = 0.0004 +	F = 32.96±57 W = 5617 p-value = 0.08 ∼
entr. 2	F = 28.05±0.94 W = 6565 p-value = 7e-05 −	F = 29.12±0.7 W = 5721 p-value = 0.05 ∼	F = 21.32±44 W = 5069 p-value = 0.81 ∼	F = 23.04±46 W = 4755 p-value = 0.39 ∼
ucb 1	F = 28.19±1.07 W = 4119 p-value = 0.025 +	F = 28.89±0.94 W = 5034 p-value = 0.93 ∼	F = 34.92±55 W = 6217 p-value = 0.001 +	F = 49.4±59 W = 5222 p-value = 0.55 ∼
ucb 2	F = 27.71±0.99 W = 2880 p-value = 7e-8 +	F = 27.98±0.92 W = 5289 p-value = 0.45 ∼	F = 73.3±73 W = 6256 p-value = 0.001 +	F = 60.4±71 W = 6475 p-value = 0.002 +
xwtr 1	F = 28.11±1 W = 5835 p-value = 0.032 −	F = 29.24±0.75 W = 5107 p-value = 0.77 ∼	F = 29.38±52 W = 4529 p-value = 0.13 ∼	F = 20.48±45 W = 5086 p-value = 0.76 ∼
xwtr 2	F = 28.15±0.95 W = 5805 p-value = 0.038 −	F = 29.30±0.65 W = 9869 p-value = 0.84 ∼	F = 28.88±50 W = 4482 p-value = 0.08 ∼	F = 21.24±47 W = 5057 p-value = 0.84 ∼

Local Search

Traces	3-trap (Maximization)		11 Multiplexer (Minimization)	
	DG	P2P	DG	P2P
entr. 1	F = 28.6±0.89 W = 4167 p-value = 0.03 +	F = 29.39±0.64 W = 3831 p-value = 0.002 +	F = 15.76±37 W = 6401 p-value = 2e-05 +	F = 15.44±34 W = 6248 p-value = 0.0002 +
entr. 2	F = 27.6±1.01 W = 7587 p-value = 6e-11 −	F = 29.6±0.51 W = 3864 p-value = 0.002 +	F = 30.93±49 W = 4457 p-value = 0.1 ∼	F = 17.36±39 W = 4936 p-value = 0.82 ∼
ucb 1	F = 28.5±0.97 W = 3318 p-value = 2e-05 +	F = 29.5±0.67 W = 2970 p-value = 9e-08 +	F = 40.4±57 W = 6002 p-value = 0.008 +	F = 40.84±52 W = 5492 p-value = 0.19 ∼
ucb 2	F = 28.5±1.06 W = 1520 p-value = 3e-18 +	F = 28.92±0.81 W = 2614 p-value = 9e-10 +	F = 64.58±62 W = 6644 p-value = 3.5e-05 +	F = 60.64±63 W = 6490 p-value = 0.0001 +
xwtr 1	F = 28.06±0.98 W = 5992 p-value = 0.01 −	F = 29.43±0.63 W = 4460 p-value = 0.15 ∼	F = 21.02±45 W = 4929 p-value = 0.81 ∼	F = 14.4±35 W = 5237 p-value = 0.4 ∼
xwtr 2	F = 28.44±0.94 W = 5034 p-value = 0.93 ∼	F = 29.58±0.6 W = 3822 p-value = 0.001 +	F = 29.36±56 W = 4461 p-value = 0.08 ∼	F = 19.92±41 W = 4918 p-value = 0.78 ∼

Table 6 Best fitness comparison between the breeding (*top*) and local search (*bottom*) policies and the respective cases if only degradation occurs (results in Table 5).

with or without using active fault-tolerance mechanisms, and "−" states that actually the given policy influences the performance in a negative way.

An overall analysis of the results shows the positive effect of using fault-tolerant policies on the algorithmic performance. A global count reveals 21 cases in which churn-aware policies outperform the respective degradation scenarios (i.e. 21 times "+"), also 21 times results are equivalent ("∼") and there are only 6 cases in which the use of fault-tolerant policies damages the search process ("−"). Such a negative effect can however be easily framed under a certain pattern: it only occurs in the DG genetic algorithm (i.e. tackling the 3-trap problem) under low churn rate traces. In fact, algorithms mostly benefit from the policies ("+") in scenarios with high churn.

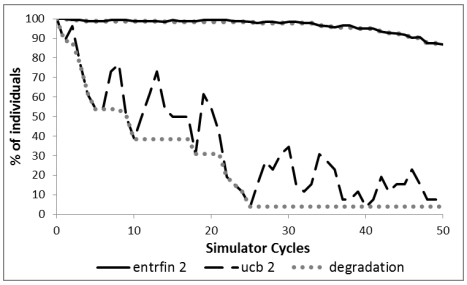


Fig. 8 Two examples of traces with a low churn rate (*entrfin 2*) and a high churn rate (*ucb 2*). Ascendant slopes represent node arrivals. In order to generate new individuals, the different fault-tolerance policies are applied while ascending.

To illustrate better what a low/high churn rate means, Fig. 8 depicts both, a low and a high churned trace. In high-churned traces, the computational efforts due to the generation of new individuals range from 20% to 40% of the total amount of evaluations (see Table 7). That is the case of *entrfin 1*, *ucb 1* and *ucb 2* traces (bold font in Table 7). On the other hand, *entrfin 2*, *xwtr 1* and *wxtr 2* are low-churned traces in which computational efforts due to node arrivals add less than 4% to the overall search process. In these low-churned scenarios fault-tolerant policies do not have a great influence on the results and in most of the cases the approaches behave as in the degradation scenario (i.e. "∼" symbols).

Taking into account previous results, the robustness of a volunteer-based evolutionary algorithms can be drawn from an intuitive perspective. On the one hand, low churn rates degrade the algorithmic performance gracefully. On the other hand, high churn rates promote the exploration of the search landscape as a variable population size strategy generates new individuals on-line. Both fault-tolerance mechanisms acting together balance the search process. Therefore, the algorithmic degradation is only subject to extreme loss of resources.

This behavior is again more relevant in the P2P approach, it shows to be more robust than the DG approach since there is not even a single case in which the variable population size strategy reduces the performance. Additionally, results are outstanding when combining the P2P approach with the *local search* policy. In fact, the trade-off represents the best found alternative to overcome churn dynamics.

Trace	FEs		Volatility (Eq. 2)		FEs due to node arrivals
failure-free	200000	90000	-	-	-
entrfin 1	**176000**	**78000**	**0.13**	**0.13**	**38% - 34%**
entrfin 2	196000	87000	0.02	0.03	3% - 0%
ucb 1	**116000**	**54000**	**0.42**	**0.40**	**31% - 27%**
ucb 2	**72000**	**42000**	**0.64**	**0.53**	**33% - 21%**
xwtr 1	176000	84000	0.13	0.06	4% - 2%
xwtr 2	180000	84000	0.10	0.06	4% - 2%
	GP	**GA**	**GP**	**GA**	

Table 7 Number of fitness evaluations (FEs) and volatility for the different traces and approaches (GA and GP) in the host-churn scenario. The sixth column shows the percentage of FEs that is employed by the new nodes at arrival. The first value corresponds to the GP problem and the second to the GA one.

Fig. 9(a) and 9(b) show how the hybridization of P2P with local search tends to perform as in the error-free case. Both graphs compile the results for the P2P approach in the different test cases: error-free, degradation and variable population size scenarios.

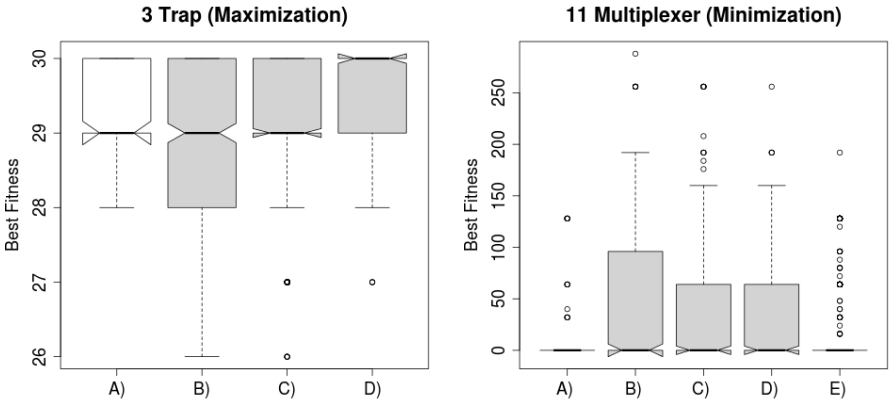
Fig. 9(a) shows the best fitness distributions for the 3-trap function where the P2P error-free run is used as a baseline for comparison. The graceful degradation in the error-prone scenario ("B") allows the algorithm to yield the same average best fitness as the error-free run. However, a higher variance of the distribution points out that the algorithm performance decreases under high rates of failure (e.g. as in *ucb 1* or *ucb 2*). The *breeding* policy ("C") has in that sense a positive effect on the performance and the algorithm is able to yield equivalent results to the error-free case. Nonetheless, the *local search* policy ("D") presents remarkable results, not only overcoming failures but also outperforming the error-free run.

With respect to the results in genetic programming (i.e. 11 Multiplexer problem), Fig. 9(b) shows a similar trend as the one presented for the genetic algorithm (3-trap): the variance of the solutions increases in the degradation scenario ("B") and improves with any of the variable population size schemes ("C" and "D"). Additionally, the algorithm can perform as in the error-free case if we only consider the worst trace (*ucb 2*) out of the compilation ("E"). That means that the P2P-GP approach is resilient to churn dynamics except for the most extreme scenario of churn.

6 Discussion and Conclusions

In this paper, we have studied different approaches for parallelizing evolutionary algorithms (EAs) in volunteer computing systems that we call *volunteer-based evolutionary algorithms*. The aim of a volunteer-based EA is to speed up execution times in demanding optimization problems via the massive scalability of volunteer platforms. However, the particularity of such infrastructures is that the computational power is aggregated from resources that users or institutions donate worldwide throughout Internet. Volunteer platforms are therefore under somebody else's control and the volatility of resources (a.k.a. host-churn) should be made explicit within the algorithm design.

To gain insight into the issue of designing robust algorithms, we have conducted an empirical experimentation on simple genetic algorithm and genetic programming approaches using two distributed EA models for creating the volunteer application:



(a) (b)

Fig. 9 Best fitness distributions in 3-trap (*a*) and 11 multiplexer (*b*) for the P2P approach. A) is the error-free run (*white*) and in *gray* the compilation of results of all the traces for B) degradation scenario, C) breeding policy and D) local search policy. E) is compiled from D but erasing the results due to the *ucb 2* trace which is the highest churned trace. The Wilcoxon test between A and E provides a $p-value = 0.053$ meaning that the hybridization of Peer-to-Peer Genetic Programming with local search is able to overcome failures in most of the churn scenarios.

the first based on centralized desktop grids (DGs) and the second on decentralized peer-to-peer systems (P2P).

In a first set of experiments, we assume idealistic conditions: volunteers provide unlimited and failure-free resources to the running experiment. The idea is to gather some first results in a best-case scenario in order to establish a baseline for comparisons. Under these conditions, the two algorithms tested here, DG and P2P, speed up linearly with respect to the number of available hosts in an ideal case, where no cost in communications is assumed. However, the good progress in fitness of both volunteer-based approaches is more relevant. The DG approach is algorithmically equal to a canonical – and eventually sequential – EA, with the evolutionary loop hosted in the master node; but the P2P approach, where the breeding scheme is decentralized, is shown to outperform the algorithmic results in the two problems under study.

Experiments are then reproduced using host-churn traces of real-world platforms in which a predetermined portion of the population is assigned to every computer: the initial population is equally divided among the available hosts. A stringent assumption is then made: nodes that fail never become available again. Therefore the population size can only shrink at run-time. In that scenario, small failure rates do not alter the quality of solutions and it is only under high failure rates – experiments finishing with roughly half of the initial population – where the optimization process is damaged. Given that a large amount of resources has to fail in order to induce losses of quality, it can be concluded that volunteer-based EAs suffer a graceful degradation. A proof for that is that the decentralized approach – losing up to 70% of the individuals – still outperforms results of the centralized approach running in a failure-free scenario.

Previous conclusions refer to an inherent fault-tolerance of EAs for degradation; however, host-churn means that not only node decay happens but also that new com-

puters join the system. In order to design fault-tolerant strategies aware of such dynamics, we propose generating new individuals at node arrivals. Thereby the population size will change depending on the number of available hosts at a given moment, with fault-tolerant mechanisms acting in both cases: graceful degradation in decreasing stages and the breeding of new individuals while increasing. Specifically, we have tested in this paper two different policies for generating individuals: the first one breeds extra individuals from the existing population which are then assigned to the incoming nodes. Additionally, the second policy clones individuals as new nodes arrive and then performs local search on them.

The use of such simple strategies has been shown to be enough to preserve – and even improve – the quality of solutions in most of the churn scenarios and problems under study, either for the centralized or the decentralized approach. Furthermore, the hybridization of the P2P approach with the local search policy has provided the most outstanding results: the volunteer system needs to lose up to 90% of the initial resources to diminish the algorithmic performance.

From these results we conclude that volunteer-based EAs are viable even in the presence of high churn rates. On the one hand, the EA relies on its own inherent fault-tolerance to bear degradation. This only mechanism has been shown sufficient to overcome small failure rates. On the other hand, to ensure that the algorithm is resilient to highly dynamical environments, we have found that the algorithm should provide strategies for utilizing new available resources; we have tested two policies for breeding new individuals in this sense.

As future lines of work, two main issues were discerned during the development of this paper. The first is related to the validation of previous results in real infrastructures where not only fault-tolerance but also heterogeneity or asynchrony arise as main issues. To that end, we plan to create a volunteer platform in student laboratories for testing and tracing time-consuming problems. The second issue is related to security and trust management. Given that the ownership of computing resources in volunteer systems is under the control of private users, malicious users may pose a threat to the optimization process and try to disrupt the algorithmic convergence. We find that an appropriate use of P2P protocols here becomes critical: on the one hand, a poorly designed protocol may end up with threats spreading virally in the system; but on the other hand, an epidemic system can also behave as a self-organized firewall, restricting the influence scope of a malicious node locally.

Acknowledgment

We would like to thank Claudia Höfer for proofreading this manuscript. This work was supported by the Luxembourg FNR Green@Cloud project (INTER/CNRS/11/03) and by the Spanish Ministry of Science Project (TIN2011-28627-C04).

References

[Ackley(1987)] Ackley DH (1987) A connectionist machine for genetic hillclimbing. Kluwer Academic Publishers, Norwell, MA, USA

[Alba and Tomassini(2002)] Alba E, Tomassini M (2002) Parallelism and evolutionary algorithms. IEEE Transactions on Evolutionary Computation 6(5):443–462

[Anderson(2004)] Anderson D (2004) BOINC: a system for public-resource computing and storage. In: Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on, pp 4–10

[Anderson et al(2002)] Anderson DP, Cobb J, Korpela E, Lebofsky M, Werthimer D (2002) SETI@home: an experiment in public-resource computing. Commun ACM 45(11):56–61, DOI http://doi.acm.org/10.1145/581571.581573

[Arenas et al(2002)] Arenas MG, Collet P, Eiben AE, Jelasity M, Merelo JJ, Paechter B, Preuß M, Schoenauer M (2002) A framework for distributed evolutionary algorithms. In: Proceedings of the 7th International Conference on Parallel Problem Solving from Nature, Springer-Verlag, London, UK, UK, PPSN VII, pp 665–675, URL http://dl.acm.org/citation.cfm?id=645826.669596

[Arenas et al(2011)] Arenas MG, Merelo JJ, Mora AM, Castillo PA, Romero G, Laredo JLJ (2011) Assessing speed-ups in commodity cloud storage services for distributed evolutionary algorithms. In: IEEE Congress on Evolutionary Computation, IEEE, pp 304–311

[Bánhelyi et al(2009)] Bánhelyi B, Biazzini M, Montresor A, Jelasity M (2009) Peer-to-peer optimization in large unreliable networks with branch-and-bound and particle swarms. In: Giacobini M, Brabazon A, Cagnoni S, Di Caro G, Ekárt A, Esparcia-Alcázar A, Farooq M, Fink A, Machado P (eds) Applications of Evolutionary Computing, Lecture Notes in Computer Science, vol 5484, Springer Berlin / Heidelberg, pp 87–92

[Biazzini and Montresor(2010)] Biazzini M, Montresor A (2010) Gossiping differential evolution: A decentralized heuristic for function optimization in p2p networks. In: ICPADS, IEEE, pp 468–475

[Biazzini and Montresor(2013)] Biazzini M, Montresor A (2013) p2poem: Function optimization in p2p networks. Peer-to-Peer Networking and Applications 6(2):213–232, DOI 10.1007/s12083-012-0152-8, URL http://dx.doi.org/10.1007/s12083-012-0152-8

[Cahon et al(2005)] Cahon S, Melab N, Talbi EG (2005) An enabling framework for parallel optimization on the computational grid. In: Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on, vol 2, pp 702–709 Vol. 2, DOI 10.1109/CCGRID.2005.1558632

[Derby et al(2013)] Derby O, Veeramachaneni K, O'Reilly UM (2013) Cloud driven design of a distributed genetic programming platform. In: Esparcia-Alcázar A (ed) Applications of Evolutionary Computation, Lecture Notes in Computer Science, vol 7835, Springer Berlin Heidelberg, pp 509–518, DOI 10.1007/978-3-642-37192-9_51, URL http://dx.doi.org/10.1007/978-3-642-37192-9_51

[Desell et al(2008)] Desell T, Szymanski B, Varela C (2008) An asynchronous hybrid genetic-simplex search for modeling the Milky Way galaxy using volunteer computing. In: Proceedings of the 10th annual conference on Genetic and Evolutionary Computation, ACM, pp 921–928

[Desell et al(2010)] Desell T, Magdon-ismail M, Szymanski B, A C, Newberg H, Anderson DP (2010) Validating evolutionary algorithms on volunteer computing grids. In: International conference on distributed applications and interoperable systems, pp 29–41

[Gagné and Parizeau(2003)] Gagné C, Parizeau M (2003) The master-slave architecture for evolutionary computations revisited. In: Proceedings of the Genetic and Evolutionary Computation Conference, Chicago, IL, pp 1578–1579

[Ghosh(2006)] Ghosh S (2006) Distributed systems: an algorithmic approach. Chapman & Hall/CRC

[Godfrey and Stoica(2005)] Godfrey PB, Stoica I (2005) Heterogeneity and load balance in distributed hash tables. In: 24th Annual Joint Conference of the IEEE Computer and Communications Societies, vol 1, pp 596–606 DOI 10.1109/INFCOM.2005.1497926

[Gonzalez and Fernández(2007)] Gonzalez DL, Fernández F (2007) Analyzing fault tolerance on parallel genetic programming by means of dynamic-size populations. In: Congress on Evolutionary Computation, Singapore, vol 1, pp 4392 – 4398

[Gonzalez et al(2009a)] Gonzalez DL, Fernández F, Casanova H (2009a) Characterizing fault tolerance in genetic programming. In: Workshop on Bio-Inspired Algorithms for Distributed Systems, Barcelona, Spain, pp 1–10

[Gonzalez et al(2009b)] Gonzalez DL, Fernández F, Trujillo L, Olague G, Araujo L, Castillo PA, Merelo JJ, Sharman K (2009b) Increasing gp computing power for free via desktop grid computing and virtualization. In: Baz DE, Spies F, Gross T (eds) PDP, IEEE Computer Society, pp 419–423

[Gonzalez et al(2010)] Gonzalez DL, Laredo JLJ, de Vega FF, Merelo JJ (2010) Characterizing fault-tolerance of genetic algorithms in desktop grid systems. In: Cowling PI, Merz P (eds) EvoCOP, Springer, Lecture Notes in Computer Science, vol 6022, pp 131–142

[Hidalgo et al(2007)] Hidalgo I, Fernández F, Lanchares J, Lombraña D (2007) Is the island model fault tolerant? In: Genetic and Evolutionary Computation Conference, London, England, vol 2, p 1519

[Jelasity and van Steen(2002)] Jelasity M, van Steen M (2002) Large-scale newscast computing on the Internet. Tech. Rep. IR-503, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, URL http://www.cs.vu.nl/pub/papers/globe/IR-503.02.pdf

[Karafotias et al(2011)] Karafotias G, Haasdijk E, Eiben AE (2011) An algorithm for distributed on-line, on-board evolutionary robotics. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation, ACM, New York, NY, USA, GECCO '11, pp 171–178, DOI http://doi.acm.org/10.1145/2001576.2001601, URL http://doi.acm.org/10.1145/2001576.2001601

[Kondo et al(2007)] Kondo D, Fedak G, Cappello F, Chien A, Casanova H (2007) Characterizing resource availability in enterprise desktop grids. Future Generation Computer Systems 23(7):888–903

[Koza(1992)] Koza JR (1992) Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA

[Laredo et al(2008a)] Laredo JLJ, Castillo PA, Mora AM, Merelo JJ (2008a) Exploring population structures for locally concurrent and massively parallel evolutionary algorithms. In: IEEE Congress on Evolutionary Computation (CEC2008), WCCI2008 Proceedings, IEEE Press, Hong Kong, pp 2610–2617

[Laredo et al(2008b)] Laredo JLJ, Castillo PA, Mora AM, Merelo JJ, Fernandes C (2008b) Resilience to churn of a peer-to-peer evolutionary algorithm. Int J High Performance Systems Architecture 1(4):260–268, DOI http://dx.doi.org/10.1504/IJHPSA.2008.024210

[Laredo et al(2010)] Laredo JLJ, Eiben AE, van Steen M, Merelo JJ (2010) Evag: a scalable peer-to-peer evolutionary algorithm. Genetic Programming and Evolvable Machines 11(2):227–246

[Laredo et al(2012)] Laredo JLJ, Bouvry P, Mostaghim S, Merelo JJ (2012) Validating a peer-to-peer evolutionary algorithm. In: et al CDC (ed) EvoApplications, Springer, Lecture Notes in Computer Science, vol 7248, pp 436–445

[Lim et al(2007)] Lim D, soon Ong Y, sung Lee A B (2007) Efficient hierarchical parallel genetic algorithms using grid computing. Future Generation Computer Systems 23(4):658–670

[Montero and Riff(2011)] Montero E, Riff MC (2011) On-the-fly calibrating strategies for evolutionary algorithms. Information Sciences 181(3):552 – 566, DOI 10.1016/j.ins.2010.09.016, URL http://www.sciencedirect.com/science/article/pii/S0020025510004561

[O'Reilly et al(2013)] O'Reilly UM, Wagy M, Hodjat B (2013) EC-Star: A Massive-Scale, Hub and Spoke, Distributed Genetic Programming System. In: R. Riolo et al. (eds.) Genetic Programming Theory and Practice X, Genetic and Evolutionary Computation, Springer Science + Business Media New York, pp 73–85

[Pospichal et al(2010)] Pospichal P, Jaros J, Schwarz J (2010) Parallel genetic algorithm on the cuda architecture. In: Di Chio C, Cagnoni S, Cotta C, Ebner M, Ekárt A, Esparcia-Alcazar A, Goh CK, Merelo J, Neri F, *Preuβ* M, Togelius J, Yannakakis G (eds) Applications of Evolutionary Computation, Lecture Notes in Computer Science, vol 6024, Springer Berlin / Heidelberg, pp 442–451

[Roy et al(2009)] Roy G, Lee H, Welch JL, Zhao Y, Pandey V, Thurston D (2009) A distributed pool architecture for genetic algorithms. In: Proceedings of the Eleventh conference on Congress on Evolutionary Computation, IEEE Press, Piscataway, NJ, USA, CEC'09, pp 1177–1184, URL http://dl.acm.org/citation.cfm?id=1689599.1689756

[Scriven et al(2008a)] Scriven I, Ireland D, Lewis A, Mostaghim S, Branke J (2008a) Asynchronous multiple objective particle swarm optimisation in unreliable distributed environments. In: IEEE Congress on Evolutionary Computation, IEEE, pp 2481–2486

[Scriven et al(2008b)] Scriven I, Lewis A, Ireland D, Lu J (2008b) Decentralised distributed multiple objective particle swarm optimisation using peer to peer networks. In: IEEE Congress on Evolutionary Computation, IEEE, pp 2925–2928

[Scriven et al(2009)] Scriven I, Lewis A, Mostaghim S (2009) Dynamic search initialisation strategies for multi-objective optimisation in peer-to-peer networks. In: IEEE Congress on Evolutionary Computation, IEEE, pp 1515–1522

[Stats(2013)] Stats (2013) Internet world stats. http://www.internetworldstats.com, accessed on March 8th, 2013.

[Steinmetz and Wehrle(2005)] Steinmetz R, Wehrle K (eds) (2005) Peer-to-Peer Systems and Applications, Lecture Notes in Computer Science, vol 3485, Springer

[Stutzbach and Rejaie(2006)] Stutzbach D, Rejaie R (2006) Understanding churn in peer-to-peer networks. In: Proceedings of the 6th ACM SIGCOMM on Internet measurement (IMC 2006), ACM Press, New York, NY, USA, pp 189–202, DOI 10.1145/1177080.1177105, URL http://portal.acm.org/citation.cfm?id=1177105

[Thierens(1999)] Thierens D (1999) Scalability problems of simple genetic algorithms. Evolutionary Computation 7(4):331–352

[Varrette et al(2008)] Varrette S, Ostaszewski M, Bouvry P (2008) Nature inspired algorithm-based fault tolerance on global computing platforms. application to symbolic regression. In: Proc. of the Intl. Conf. on Metaheuristics and Nature Inspired Computing (META'08), Hammamet, Tunisia, URL http://www2.lifl.fr/META08/

[Voulgaris et al(2004)] Voulgaris S, Jelasity M, van Steen M (2004) Agents and Peer-to-Peer Computing, Lecture Notes in Computer Science (LNCS), vol 2872, Springer Berlin / Heidelberg, chap A Robust and Scalable Peer-to-Peer Gossiping Protocol, pp 47–58. DOI 10.1007/b104265

[Wickramasinghe et al(2007)] Wickramasinghe WRMUK, van Steen M, Eiben AE (2007) Peer-to-peer evolutionary algorithms with adaptive autonomous selection. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation, ACM, New York, NY, USA, GECCO '07, pp 1460–1467, DOI http://doi.acm.org/10.1145/1276958.1277225, URL http://doi.acm.org/10.1145/1276958.1277225

[Zhao et al(2009)] Zhao Z, Yang F, Xu Y (2009) Ppvc: A p2p volunteer computing system. In: 2nd IEEE International Conference on Computer Science and Information Technology, IEEE, pp 51–55