The human in the loop: volunteer-based metacomputers as a socio-technical system

Juan-J. Merelo*[1], Paloma de las Cuevas [1], Pablo García-Sánchez [1], Mario García-Valdez[2]

[1]Dept. of Computer Architecture and Technology and CITIC University of Granada
[2]Dept. of Graduate Studies at Instituto Tecnológico de Tijuana
jmerelo@ugr.es, mario@tectijuana.edu.mx

Abstract

Volunteer computing is a form of distributed computing where users decide on their participation and the amount of time and other resources they will "lend". This makes them an essential part of the algorithm and of the performance of the whole system. As a socio-technical system, this participation follows some patterns and in this paper we examine the result of several volunteer distributed evolutionary computation experiments and try to find out what those patterns are and what makes an experiment successful or not, including the feedback loop that is created between the users and the algorithm itself.

Introduction

Some time ago, our group faced the problem of diminishing funds for buying new hardware. This was aggravated by the increasing maintenance costs and extended downtime resulting from the continuous failures of existing clusters. Considering this, we leveraged our experience in the design of web applications with JavaScript and other volunteer and unconventional distributed evolutionary computing systems to design and release a new free framework that would allow anyone to create a volunteer distributed evolutionary computation (EC) experiment using cloud resources as servers and browsers as clients. This framework was called NodIO (Merelo et al., 2016). NodIO provides server infrastructure for volunteer-based distributed evolutionary computing experiments by providing a chromosome *pool*. This pool is used by clients in browsers and any other using the application programming interface (API) to put chromosomes and retrieve them, working then as a loose, asynchronous and *ad hoc* connection among all clients using it.

This loose connection provides a low-overhead way to connect desktop experiments with volunteer-based ones, with all of them contributing to the pool, but every one of them working as separate island carrying out their own evolutionary algorithms. This is why NodIO is proposed mainly as a *complement* to existing resources such as desktop systems or laptops. As long as it provides a non-null computational capability that can help existing resources find the solution faster it will have found its purpose. The main use case is someone setting up NodIO in the cloud, writing a fitness evaluation function and running a client from his or her own computer, but requesting help in social networks for additional resources. That is why, in this system, we have considered the whole *social* aspect in the design, with issues related to security, trust and privacy among others. The computing system becomes a *socio-technical system* (Vespignani et al., 2009). In this paper, we are going to focus on measuring the response of users to experiments, that is, the time they spend running it, but at the same time we will also focus on the technical aspects of the server and how these might change the behavior of users, improving the capability of the system.

Our research group is committed to open science, and we think this is a very important part of the techno-social system. By being transparent, incentives to cheat are reduced and, in fact, we have detected no issue for the time being. Next we present the state of the art in web-based volunteer computing systems along with attempts to predict and model its behavior.

State of the art

Volunteer computing involves a user running a program voluntarily and, as such, has been deployed in many different ways from the beginning of the Internet, starting with the SETI@home framework for processing extraterrestrial signals (Anderson et al., 2002). However the dual introduction of JavaScript as a universal language for the browser and the browser as an ubiquitous web and Internet client has made this combination the most popular for volunteer computing frameworks such as the one we are using here, and whose first version was described in (Merelo-Guervós and García-Sánchez, 2015).

JavaScript can be used for either unwitting (Klein and Spector, 2007; Boldrin et al., 2007) or volunteer (Langdon, 2005; Merelo et al., 2007) distributed evolutionary computation and it has been used ever since by several authors, including more recent efforts (Desell et al., 2008; Duda and Dłubacz, 2013; Gonzalez et al., 2008). Many other researchers have used Java (Chong and Langdon, 1999); oth-

Downloaded from http://direct.mit.edu/isal/proceedings-pdf/alif2016/28/648/2325851/978-0-262-33936-0-ch103.pdf by guest on 23 May 2025

ers have embraced peer to peer systems (Jin et al., 2006; Wang and Xu, 2008; Merelo-Guervós et al., 2012). These computing platforms avoid single points of failure, the server, and once installed need no effort to gather new users for an experiment, but the cost of acquiring new users is high since they need to be set up.

The number of users is key in the performance of these systems, but it also essential to adapt the algorithm itself to the available resources, as shown by (Milani, 2004), although EAs can be readily distributed via population splitting or by farming out the evaluation to all the nodes available. However, user churn affects experiment performance (González Lombraña et al., 2010; Nogueras and Cotta, 2015) and also the performance of the algorithm itself (Laredo et al., 2014). All these issues imply that a the performance of a volunteer system cannot be measured without first understanding its dynamics. Initial work was done for peer to peer systems by Stutzbach et al. (Stutzbach and Rejaie, 2006) and extended to volunteer computing by Laredo et al. (Laredo et al., 2008a,b). A similar study was performed by Martnez et al. on the Capataz system (Martınez and Val, 2015); however, in this case the number of computers used was known in advance and the main focus was on measuring the speed up and how job bundling helped to reduce overhead and enhance performance. On the contrary, in this paper, we will use *actual* volunteers.

Some of the essential metrics in volunteer computing like the number of users or the time spent by every one in the computation in browser-based volunteer computing experiments, have only been studied in a limited way in (Laredo et al., 2014) on the basis of a single run. Studies using volunteer computing platforms such as SETI@home (Javadi et al., 2009; Merelo et al., 2008) found out that the Weibull, log-normal and Gamma distribution modeled quite well the availability of resources in several clusters of that framework; the shape of those distributions is a skewed bell with more resources in the *low* areas than in the high areas: there are many users that give a small amount of cycles, while there are just a few that give many cycles.

As far as we know, this paper presents one of the few experiments that measure the performance of a socio-technical metacomputer, that is, a spontaneously created parallel computer that uses social networks for operations such as gathering new users. Apolónia et al. (Apolónia et al., 2012) used the Facebook protocol to distribute tasks among the *walls* of friends, explicitly using the social network for computing. However, it stopped short of relating performance to the macro measures of the users' social networks. As in the previous example, a social network was used to get new network nodes; in the previous case a web page was used, while Facebook's wall was used here.

In our case, social networks are an integral part of the system and used to spontaneously obtain users. The algorithms used, as well as the methodology for gathering resources will be described next, together with the results obtained in this initial setup.

Description of the framework

In general, a distributed volunteer-based evolutionary computation system based on the browser is simply a client-server system whose client is, or it can be, embedded in the browser via JavaScript. Since JavaScript is the only language that is present across all browsers, the choice was quite clear. We should emphasize that NodIO is more intended as an auxiliary computing engine, more than the main one, so performance of JavaScript as a language is not so important; even so, we have made a comparison between JavaScript and other languages (Merelo et al., 2015) that shows that the performance of JavaScript is comparable to other interpreted languages; compiled languages would be faster, but, of course, it is impossible to gather volunteers spontaneously and without any installation with them.

In this sense, in this paper we propose the NodIO framework, a cloud or bare metal based volunteer evolutionary computing system derived from the NodEO library, whose architecture has been developed using JavaScript on the client as well as the server. All parts of the framework are free and available with a free license from `https://github.com/JJ/splash-volunteer`.

Thus, NodIO architecture has two tiers:

1. A REST server, that is, a server that includes several *routes* that can be called for storing and retrieving information (the 'CRUD" cycle: create, request, update, and delete) from the server. A JSON data format is used for the communication between clients and the server. There are two kinds of information: *problem* based, that is, related to the evolutionary algorithm such as `PUT`ing a chromosome in or `GET`ing a random chromosome from it, and *information* related to the performance and state of the experiments. It also performs logging duties, but they are basically a very lightweight and high performance data storage (Merelo, 2015). The server has the capability to run a single experiment, storing the chromosomes in a key-value store that is reset when the solution is found. This store can hold every chromosome in a particular experiment, or have a finite size that erases the oldest chromosomes once it has filled to capacity, acting as a cache. In this paper we will test both implementations.

2. A client that includes the evolutionary algorithm as JavaScript code embedded in a web page that displays graphs, some additional links, and information on the experiment. This code runs an evolutionary algorithm *island* that starts with a random population, then after every 100 generations, it sends the best individual back to the server (via a `PUT` request), and then requests a random individual back from the server (via a `GET` request). We have kept the number of generations between migrations fixed

Downloaded from http://direct.mit.edu/isal/proceedings-pdf/alif2016/28/648/2325851/978-0-262-33936-0-ch103.pdf by guest on 23 May 2025

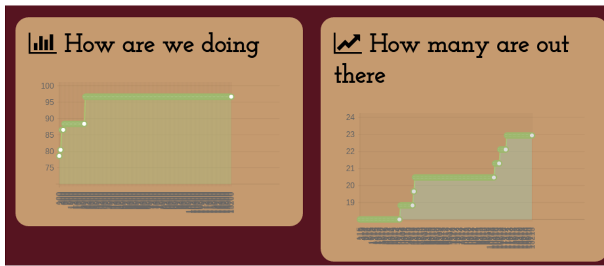

Figure 1: Screenshot showing the fitness and number of users as seen by volunteers. The one on the left shows fitness, on the right the accumulated number of users.

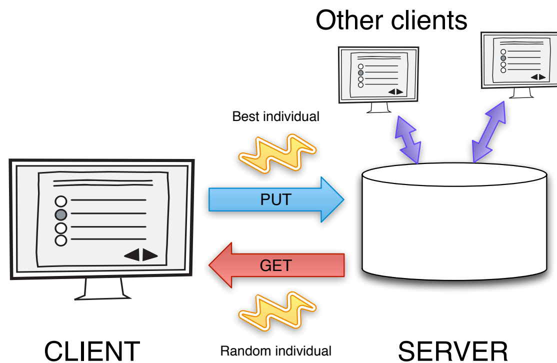

Figure 2: Description of the proposed system. Clients execute a JavaScript EA in the browser, which, every 100 generations, sends the best individual and receives a random one back from the server.

since it is a way of finding out how much real work every client has done.

Figure 2 describes the general system architecture and algorithm behavior. Different web technologies, such as JQuery and `Chart.js` have been used to build the user interface elements of the framework, a part of which is shown in Figure 1 and should be running in `http://nodio-jmerelo.rhcloud.com`.

JavaScript is a functional language, so in order to work with a problem, a fitness function must be supplied at the creation of the algorithm object, called `Classic`. In this case the classical Trap function (Ackley, 1987) has been used. Depending on the problem additional handing functions and configurations can be supplied.

The next Section will describe experiments performed to establish a baseline performance and gather initial performance results. In the first set of experiments, performed last year, we used the 40-trap problem, while the current experiments changed to the more difficult 50-trap in order to compare the performance with a more computationally intensive problem.

Modeling the performance of a volunteer-based distributed computer

Table 1: Experiment table, with summary of results.

Experiment	#Runs	Different IPs	Traps
April 4th 4/4	57	191	40
April 24th 4/24	231	559	40
July 31th 7/31	97	179	40
February, cache=128	61	75	50
February, cache=64	61	220	50
February, cache=32	39	86	50

Initial experiments were set up using the OpenShift PaaS `www.openshift.com`, which provides a free tier, making the whole experiment cost equal to $0.00. Experiments were announced through a series of posts on Twitter and, in the latest case, Telegram, and results were published in(Merelo-Guervós and García-Sánchez, 2015). For the purpose of this paper, we repeated the announcement several times through the month of February of the next year. All in all, we have the set of runs with the characteristics shown in Table 1. In general, every experiment took several days. No particular care was taken about the time of the announcement or the particular wording. Every *experiment* consisted in running until the solution of the 50-trap problem was found. When the correct solution was sent to the server, the counter was updated and the pool of solutions resets to the void set. There was no special intention to wait until all clients had finished, thus it might happen that. In fact, the islands running in the browser *spill* from one experiment to the next. If an individual that is close or even at optimum value is stored in the cache the next experiment will be affected, and it will require less time to finish. This problem has been addressed in later versions of the system, however, time to solution is not the important measure here, where we are concerned with the computational power of the system established by the number of volunteers.

The table 1shows that every experiment included more than 30 runs. The number of different IPs intervening in them varied from more than one hundred to more than five hundred in the second experiment, with a number around 50 in the second, and most recent, batch of experiments.

A summary of the results of each run is also shown in Table 2, which shows the median number of IPs intervening in each experiment, median time needed to finish the experiment, median number of HTTP `PUT`s per IP. The first striking result is that in all cases, 50% of the experiments involved 5 or less IPs. This is consistent with previous results (Merelo-Guervós and García-Sánchez, 2015) which found 6 to be the expected number of volunteer IPs. The maximum number of different IPs for each experiment is also in the same range and of the order of 10, which is also consistent

Downloaded from http://direct.mit.edu/isal/proceedings-pdf/alif2016/28/648/2325851/978-0-262-33936-0-ch103.pdf by guest on 23 May 2025

Table 2: Summary of time per run, number of IPs and number of PUTs per IP in the initial runs.

Experiment	IPs Median	IPs Max	Median time (s)	#PUTs
4/4	5	16	2040	18
4/24	5	29	732	11
7/31	5	14	260	23
Cache=128	5	17	222.2	124
Cache=64	8	38	51.3	100
Cache=32	6	19	58.9	45

with prior work and does not vary across the two different batches.

We will have to analyze differently the median time, since the two batches are solving different problems. In both cases it possesses a big range of variation, but 50% of the time takes less than several minutes, from around 4 minutes in the best case to roughly 2/3 of an hour in the worst case. Remarkably enough, the time is more consistent in the second batch and always around one minute, in two cases even less, and that happens when the median number of IPs is higher. It should be noted that while the first batch of experiments took several days in each case, the second only lasted for a few hours, with a more continued effort of publicizing it in social networks. This is specially true in the case of cache equal to 64, which is noticed by the high number of volunteers participating in the experiment. The conclusion is that, in general, the key factor in the time needed to find the solution is, as expected, the number of volunteers it is able to gather on a short notice.

The number of PUTs, every one corresponding to 100 generations, is the algorithmic result. It is relatively unchanged for the first batch and around 20, that is, 2000 generations or 2000*128 = 256000 evaluations. In this case, an "unlimited" cache was used, with all individuals sent from clients stored until the end of the experiment. However, we were interested in measuring also the performance of the algorithm itself by changing the cache size, after making it limited. As it can be seen in the table, there is a clear change in the number of evaluations needed, with smaller cache sizes producing solutions in less evaluations, until it is for the cache size = 32 roughly twice as much as with the previous problem, with 40 traps. This is a good result and is also algorithmically consistent with other results obtained using the same type of problems. Since in this paper we were interested in leveraging the user's CPU cycles by improving the algorithm, a good conclusion of this paper is that having a small pool size helps clients to obtain "good" individuals from the pool, as opposed to any individual that could be obtained before. Besides, the cache policy deletes the oldest individuals, which makes those in the pool be *current*, helping then newcomers and any participant obtain the best individuals found in the last part of the experiment. Besides, a limited cache helps also in cases with a bigger search space or longer running times when the server simply crashed due to lack of RAM.

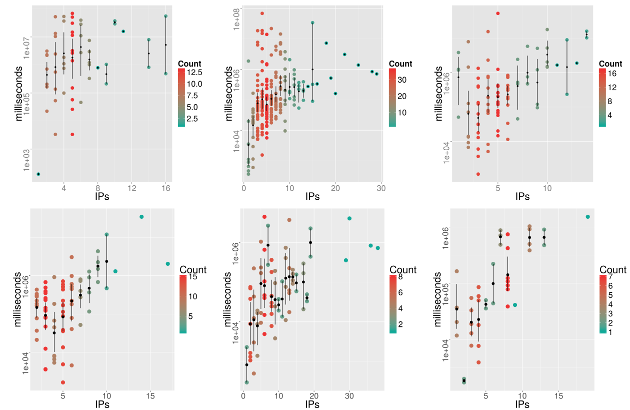

Figure 3: Duration of experiments vs. number of different IPs (nodes) participating in it, with averages per number of IPs and standard deviation shown as red dots; in the case there is a single red dot, there was a single experiment in which that many computers participated (for instance, 16 computers in the experiment in the far left or 29 in the middle one). Shades of blue indicate how many experiments included that many unique IPs, so lighter shade for a column of dots indicates that a particular number of computers happened less frequently, while darker shadow means more. From left to right and top to bottom experiments 4/4, 4/24 and 7/31, followed by experiments with 50 traps, cache=128, 64, 32.

We will have to analyze experimental data a bit further to find out why this happens and also if there are some patterns in the three sets of experiments. An interesting question to ask, for instance, is if by adding more computers it makes the experiment take less time. In fact, as shown in Figure 3, the *addition* of more computers does not seem to contribute to decreasing the time needed to finish the experiment. However, the cause-effect relationship is not clear at all. It might be the opposite: since experiments take longer to finish and might in fact be abandoned with no one contributing for some time, the probability of someone new joining them is higher. In fact, with experiments taking a few seconds and due to the way the experiments are announced, it is quite difficult that several volunteers join in in such a short period of time, even more if we take into account that volunteers are not *carried over* from previous experiments.

That is why we used a more difficult problem in the second batch of experiments, which is shown in the bottom row of Figure 3. The pattern is remarkably similar, showing a positive correlation between the time for solving the problem and the number of computers, at least for cache sizes

Downloaded from http://direct.mit.edu/isal/proceedings-pdf/alif2016/28/648/2325851/978-0-262-33936-0-ch103.pdf by guest on 23 May 2025

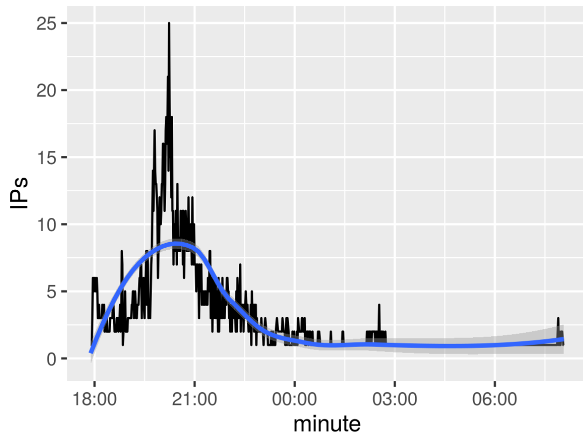

Figure 4: Simultaneous IPs every minute of the experiment with cache = 64.

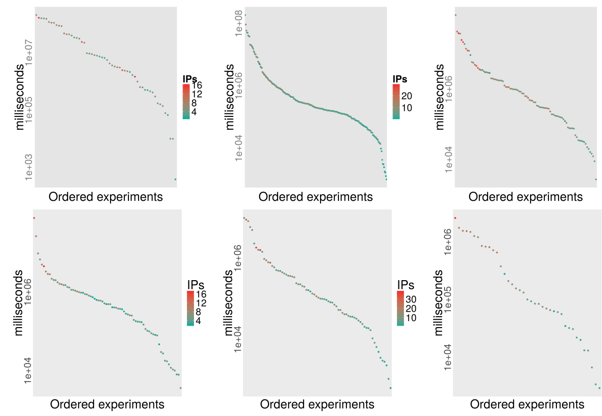

Figure 5: Duration of experiments vs. rank of experiments sorted by descending duration, with y axis in a logarithmic scale. Dot color is related to the number of IPs participating in the experiment. From left to right and top to bottom, experiments 4/4, 4/24 and 7/31 and caches=128, 64, 32.

128 and 64. However, it is interesting to observe that, for cache=32, the time needed to find the solution decreases from one to approximately 4-5 nodes, to then increase for a higher number of participating computers, distinguished by IP. The green dot at the bottom is probably an outlier that we will try to explain later on. This leads us to conclude that a larger amount of computers might contribute to speed up the solution, if the time the experiment ideally takes is sufficient, that is, of the order of a minute, and enough volunteers concur simultaneously. This is also observed, not so clearly, in the case of cache=64, with an interval of around 10 IPs obtaining less time than experiments with less or more IPs, and of the same order, between 10 and 100 seconds. If we look at the graph that shows the number of IPs or volunteers per minute for this experiment, shown in Figure 4, we see that there are peaks of more than 25 volunteers, and a period of several hours with a minimum of 6 computers and peaks of more than 10. The long period after midnight where there is a single volunteer left masks the success achieved during this set of experiments, from which we draw two lessons: first, you need a social network influencer to announce your experiments and second, no matter what, do not do any experiment after midnight. This statement, which might seem tongue in cheek, in fact, it is a conclusion drawn from the experimental data and to what extent the social network is an essential part of the description and performance of the NodIO volunteer computing system.

It is also interesting to check the distribution of the experiment duration, shown in Figure 5 and which roughly follows a Zipf's law, with similar distribution along all three runs. The 4/24 run is the most complete and shows an S-shape, which implies an accumulation of experiments taking similar time and around 100 seconds; this S-shape appears too in the experiments with cache=128 (bottom row, left). The most interesting part is the *tail*, which shows how many experiments took a desirable amount of time, on the order of 10 seconds, and which appears in all three graphs. As it can be seen, it sharply drops implying there are just a few of them, and with diminishing probability as time decreases. However, since they have a greenish color, implying a low number of IPs, they might be due to clients *carrying over* from the previous one. This is a characteristic of this implementation which will be examined later on, but at any rate, if we discard those experiments that take too much or too little, there is a decreasing exponential distribution that corresponds to the Zipf's law.

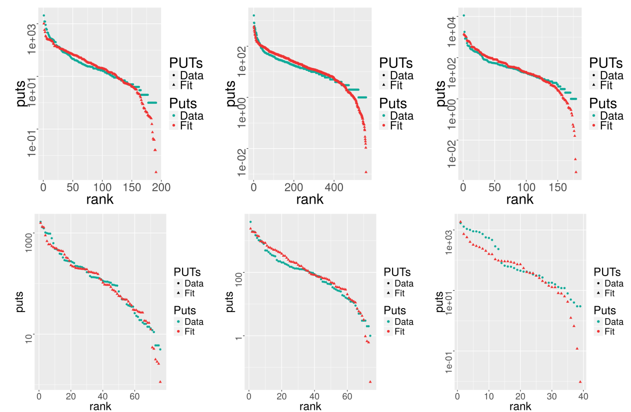

Figure 6: Number of PUTs per unique IP and fit to a Weibull distribution (in red); axis x shows IPs sorted by descending number of PUTs. From left to right and top to bottom, experiments 4/4, 4/24 and 7/31 and new experiments with cache=128, 64, 32.

A similar exponential distribution also appears if we rank HTTP PUTs, equivalents to the number of generations di-

Downloaded from http://direct.mit.edu/isal/proceedings-pdf/alif2016/28/648/2325851/978-0-262-33936-0-ch103.pdf by guest on 23 May 2025

Table 3: Weibull distribution parameters of the fit of the number of PUTs per unique IP.

Experiment	Scale σ	Shape ξ
4/4	43.07 ± 5.80	0.57 ± 0.03
4/24	22.97 ± 1.57	0.66 ± 0.02
7/31	53.18 ± 7.77	0.54 ± 0.03
Cache 128	205.28 ± 32.10	0.77 ± 0.07
Cache 64	178.99 ± 36.44	0.60 ± 0.05
Cache 32	168.15 ± 49.28	0.57 ± 0.07

vided by 100, or to evaluations divided by 12800, contributed by every user, which is shown in Figure 6. These results show a Zipf-like behavior, that is, a power law with a small *bump* in the lowest values. After testing the Generalized Extreme Value distribution and failing for the new batch of experiments, we have fitted it to a Weibull distribution Thoman et al. (1969) with the resulting parameters shown in Table 3. The inverse Weibull distribution is a special case of the GEV distribution in those papers, and appears usually in natural sciences and artificial life, usually related to decay. It has been frequently fitted to volunteer computing frameworks such as SETI@home (Javadi et al., 2009). The model that user behavior follows can be explained straighforwardly: when users visit the page, it draws their attention for a limited amount of time. They give it a chance for a few seconds. If something there amuses them or they can engage in a conversation about it, they stay for a while longer, otherwise, they leave. The *scale* parameter, which is around 20-40 in the first batch for the 40 Trap problem and between 160 and 200 in the 60 traps problem, depends mainly on the maximum number of generations people leave it running. Since it finishes or stalls after a number of generations shown in Figure 2, volunteers just leave after that. Curiously enough, the scale parameter is roughly twice the median number of PUTs per experiment, showing that, on average, the most loyal users reload the page twice after finishing or after seeing the evolution does not progress. This rule of thumb breaks down with the last experiment, however, which is interesting by itself, too.

The slope or shape parameter, on the other hand, indicates the overall shape of the curve. A value less than 1 indicates a concave (in a non-algorithmic scale) curve, with figures closer to one indicating a smaller slope. In all cases values are between 0.54 and 0.77, independently of the experiment. It might be the case that this number depends more on the total number of experiments carried out, with sets with more experiments, both in the middle, having values between 0.60 and 0.70. The distribution is remarkably similar which gives us a model of user behavior that is, to a certain extent, independent of the experiment.

These experiments show that, as it was proved for other volunteer computer frameworks and also in the case of games, user engagement follows a Weibull distribution. This makes engagement the key for leveraging the performance of the socio-technical metacomputer and a way to improve results in the future.

Conclusion

Our intention in this paper was to assess the capabilities of a socio-technical system formed by a client-server web-based framework running a distributed evolutionary algorithms and the volunteers that participate in the experiment. These volunteers are *in the cloud*, that is, available as *CPU as a service* for the persons running the experiment. In this paper we have tried to put some figures on the real size of that *cloud* and how it can be used standalone if there is no alternative, or, if other computing resources are available, in conjunction with other local or cloud-based methods to add computing power in a seamless way through the pool that NodIO creates.

After running the experiment on the 40-trap problem whose running time could be as low as a few seconds, we switched to another batch of experiments where we used the 50-trap problem and also to a pool of limited, and dwindling through the three experiments, size. Since our initial results indicated that what happened on the screen, a flat graph with no improvements or the experiment finished, influenced the amount of time that the users devoted to the experiment, a longer one could yield different results and, at the same time, result in a big pool that might either crash the server or return useless individuals to the volunteers. These new experiments have proved that using the limited pool is beneficial to finding the solution, since less evaluations are needed, but also that since the problem is more difficult, the users stay for longer in the web page, making less ephemeral the socio-technical computing system created by the simulation.

The second objective of this paper was to model the user behavior in a first attempt to try and predict performance. As should be expected, the model depends on the implementation, with contributions following a Weibull distribution, which reflects the fact that volunteer computing follows a model quite similar to that found for games or other online activities. The reverse might be true: if we want to have returning users for the experiments, it is probable that we should *gamify* the experience so that once they've done it once, they might do it more times. In the spirit of Open Science, this gamification might involve computing in real time data such as the one presented in this paper and showing it in the same page or presenting user results alongside others.

In general, linking and finding correlations between user choices and performance is an interesting avenue to explore in the future. Even if these experiments were published in a similar way, one obtained up to five times more total cycles than the one with the least number of cycles. It is also essential to obtain volunteers as fast and simultaneously as possible, so it is possible that the features of the social network in

Downloaded from http://direct.mit.edu/isal/proceedings-pdf/alif2016/28/648/2325851/978-0-262-33936-0-ch103.pdf by guest on 23 May 2025

terms of real-time use will also play a big role; synchronous webs such as Snapchat, which mystifies the writers of this paper, and Twitter, thanks to its real time nature, might be better suited than Facebook, LinkedIn or Google plus. Even as it is difficult to create controlled experiments in this area, it is an interesting challenge to explore in the future.

The other area to explore is the algorithmic area itself. Are there ways to change the evolutionary algorithm, or its visualization, so that the user has a bigger impact on the result? One of the users in Twitter even suggested to embed videos so that people spent time looking at them, but other possible way was to make the user engage the algorithm by giving him or her buttons to change the mutation rate when the algorithm is stalled, for instance. If this is combined with a score board where local performance is compared to other users, engagement might be increased and thus the performance of the system. In general there are many issues with the evolutionary algorithm implementation itself, including using different, or adaptive, policies for inserting and sending individuals to the pool, using different policies for population initialization, and also the incorporation of high-speed local resources to the pool to check what would be the real influence of the volunteer pool to the final performance.

Finally, the implementation needs some refinement in terms of programming and also ease of use. Tools such as Yeoman for generating easily new experiments might be used, so that the user would have to create only a fitness function, with the rest of the framework wrapped around automatically.

All these avenues of experimentation will be done openly following the Open Science policy of our group, which, in fact, contributes to establish trust and security between us and volunteers and is an essential feature of the system. That is why this paper, as well as the data and processing scripts, are published with a free license in GitHub at `https://github.com/JJ/modeling-volunteer-computing`.

Acknowledgments

This work has been supported in part by TIN2014-56494-C4-3-P (Spanish Ministry of Economy and Competitivity), PROY-PP2015-06 (Plan Propio 2015 UGR) . We would also like to thank the anonymous reviewers of previous versions of this paper who have really helped us to improve this paper (and our work) with their suggestions. We are also grateful to Anna Sáez de Tejada for her help with the data processing scripts. We are also grateful to `@otisdriftwood` for his help gathering users for the new experiments.

References

Ackley, D. H. (1987). *A connectionist machine for genetic hillclimbing*. Kluwer Academic Publishers, Norwell, MA, USA.

Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., and Werthimer, D. (2002). SETI@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61.

Apolónia, N., Ferreira, P., and Veiga, L. (2012). Enhancing online communities with cycle-sharing for social networks. In *Computational Social Networks*, pages 161–195. Springer.

Boldrin, F., Taddia, C., and Mazzini, G. (2007). Distributed computing through web browser. In *Vehicular Technology Conference, 2007. VTC-2007 Fall. 2007 IEEE 66th*, pages 2020–2024. IEEE.

Chong, F. S. and Langdon, W. B. (1999). Java based distributed Genetic Programming on the internet. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, page 1229, Orlando, Florida, USA. Morgan Kaufmann. Full text in technical report CSRP-99-7.

Desell, T., Szymanski, B., and 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*, GECCO ’08, pages 921–928, New York, NY, USA. ACM.

Duda, J. and Dłubacz, W. (2013). Distributed evolutionary computing system based on web browsers with JavaScript. In *Applied Parallel and Scientific Computing*, pages 183–191. Springer.

Gonzalez, D. L., de Vega, F. F., Trujillo, L., Olague, G., de la O, F. C., Cardenas, M., Araujo, L., Castillo, P. A., and Sharman, K. (2008). Increasing GP computing power via volunteer computing. *CoRR*, abs/0801.1210.

González Lombraña, D., Laredo, J. L. J., Fernández de Vega, F., and Merelo Guervós, J. J. (2010). Characterizing fault-tolerance of genetic algorithms in desktop grid systems. In *Evolutionary Computation in Combinatorial Optimization*, pages 131–142. Springer.

Javadi, B., Kondo, D., Vincent, J.-M., and Anderson, D. P. (2009). Mining for statistical models of availability in large-scale distributed systems: An empirical study of SETI@home. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009. MASCOTS’09. IEEE International Symposium on*, pages 1–10. IEEE.

Jin, H., Luo, F., Liao, X., Zhang, Q., and Zhang, H. (2006). Constructing a P2P-based high performance

Downloaded from http://direct.mit.edu/isal/proceedings-pdf/alif2016/28/648/2325851/978-0-262-33936-0-ch103.pdf by guest on 23 May 2025

computing platform. In *2006 International Workshop on P2P for High Performance Computational Sciences (P2P-HPCS06)*, volume 3994 of *LECTURE NOTES IN COMPUTER SCIENCE*, pages 380–387. Springer.

Klein, J. and Spector, L. (2007). Unwitting distributed genetic programming via asynchronous JavaScript and XML. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1628–1635, New York, NY, USA. ACM.

Langdon, W. B. (2005). Pfeiffer – A distributed open-ended evolutionary system. In Edmonds, B., Gilbert, N., Gustafson, S., Hales, D., and Krasnogor, N., editors, *AISB'05: Proceedings of the Joint Symposium on Socially Inspired Computing (METAS 2005)*, pages 7–13, University of Hertfordshire, Hatfield, UK. SSAISB 2005 Convention.

Laredo, J. L. J., Bouvry, P., González, D. L., de Vega, F. F., García-Arenas, M., Merelo-Guervós, J. J., and Fernandes, C. M. (2014). Designing robust volunteer-based evolutionary algorithms. *Genetic Programming and Evolvable Machines*, 15(3):221–244.

Laredo, J. L. J., Castillo, P. A., Mora, A. M., Fernandes, C., and Merelo, J. J. (2008a). Addressing Churn in a Peer-to-Peer Evolutionary Algorithm. In *WPABA'08 - First International Workshop on Parallel Architectures and Bioinspired Algorithms, Toronto, Canada*, pages 5–12. Complutense University Of Madrid.

Laredo, J. L. J., Castillo, P. A., Mora, A. M., Fernandes, C. M., and Merelo, J. J. (2008b). Resilience to churn of a peer-to-peer evolutionary algorithm. *International Journal of High Performance Systems Architecture*, 1(4):260–268.

Martınez, G. J. and Val, L. (2015). Capataz: a framework for distributing algorithms via the World Wide Web. *CLEI Electronic Journal*, 18(2):1.

Merelo, J. J. (2015). Low or no cost distributed evolutionary computation. In Camacho, D., Braubach, L., Venticinque, S., and Badica, C., editors, *Intelligent Distributed Computing VIII*, volume 570 of *Studies in Computational Intelligence*, pages 3–4. Springer International Publishing.

Merelo, J. J., Castillo, P., Laredo, J., Mora, A., and Prieto, A. (2008). Asynchronous distributed genetic algorithms with JavaScript and JSON. In *WCCI 2008 Proceedings*, pages 1372–1379. IEEE Press.

Merelo, J. J., García, A. M., Laredo, J. L. J., Lupión, J., and Tricas, F. (2007). Browser-based distributed evolutionary computation: performance and scaling behavior. In *GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, pages 2851–2858, New York, NY, USA. ACM Press.

Merelo, J.-J., García-Sánchez, P., García-Valdez, M., and Blancas, I. (2015). There is no fast lunch: an examination of the running speed of evolutionary algorithms in several languages. *ArXiv e-prints*.

Merelo, J.-J., García-Valdez, M., Castillo, P. A., García-Sánchez, P., de las Cuevas, P., and Rico, N. (2016). NodIO, a JavaScript framework for volunteer-based evolutionary algorithms : first results. *ArXiv e-prints*.

Merelo-Guervós, J.-J. and García-Sánchez, P. (2015). Designing and modeling a browser-based distributed evolutionary computation system. In Laredo, J. L. J., Silva, S., and Esparcia-Alcázar, A. I., editors, *Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015, Companion Material Proceedings*, pages 1117–1124. ACM.

Merelo-Guervós, J.-J., Mora, A.-M., Fernandes, C.-M., Esparcia-Alcázar, A.-I., and Jiménez-Laredo, J.-L. (2012). Pool vs. island based evolutionary algorithms: An initial exploration. In Xhafa, F., Barolli, L., and Li, K. F., editors, *3PGCIC*, pages 19–24. IEEE.

Milani, A. (2004). Online genetic algorithms. Technical report, Institute of Information Theories and Applications FOI ITHEA.

Nogueras, R. and Cotta, C. (2015). Studying fault-tolerance in island-based evolutionary and multimemetic algorithms. *Journal of Grid Computing*, pages 1–24.

Stutzbach, D. and Rejaie, R. (2006). Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 189–202. ACM.

Thoman, D. R., Bain, L. J., and Antle, C. E. (1969). Inferences on the parameters of the weibull distribution. *Technometrics*, 11(3):445–460.

Vespignani, A. et al. (2009). Predicting the behavior of techno-social systems. *Science*, 325(5939):425.

Wang, X. and Xu, S. (2008). P2HP: Construction of a cooperative server group based volunteer computing environment. In *International Conference on Internet Computing in Science and Engineering*, volume 0, pages 389–395, Los Alamitos, CA, USA. IEEE Computer Society.

Downloaded from http://direct.mit.edu/isal/proceedings-pdf/alif2016/28/648/2325851/978-0-262-33936-0-ch103.pdf by guest on 23 May 2025