Proceedings of the 2016 Winter Simulation Conference
T. M. K. Roeder, P. I. Frazier, R. Szechtman, and E. Zhou, eds.

SIMULATING AND OPTIMIZING RESOURCE ALLOCATION IN A MICRO-BLOGGING APPLICACION

Xavier Serra
Jesica de Armas
Joan Manuel Marquès

Computer Science, Multimedia and Telecommunications Department
Universitat Oberta de Catalunya
156 Rambla del Poblenou
Barcelona, SPAIN

ABSTRACT

In Volunteer Computing resources are provided by the own users, instead of by a single institution. One of its drawbacks is the unreliability of the provided resources, so their selection becomes a main point. In this paper we deal with the suitable selection of resources considering this kind of Volunteer Computing system. As the resources choice may be needed in a reduced amount of time, we cannot make use of the most powerful optimization algorithms in literature due to the time that they need to provide a solution. Instead, we propose a simple heuristic yet capable of obtaining quality results in an extremely fast way. This heuristic uses a weight system to determine each resource quality and a biased random procedure to select them accordingly. In order to tune and test it, a simulation environment of a real micro-blogging application has been developed, so that we can obtain reliable results.

1 INTRODUCTION

In recent years, *Volunteer Computing* (Anderson and Fedak 2004, Anderson. 2006) has become increasingly popular. The reasons for this popularity are easy to see: no heavy resources are required, because the computational load is shared among a large number of users. It does have, however, some drawbacks, as these resources suffer from a lack of reliability. They are provided by particulars in a voluntary way, which means that, if not handed properly, information stored there may be lost, or be delivered too slow. In order to prevent this as much as possible, information is usually replicated among different computers, henceforth nodes. One of the main issues in this kind of systems consists in selecting the right nodes in which to store each piece of information, as replicating the content of a fallen node to a new one is usually costly, and specially if it is done through the Internet. Therefore, we should choose them in such a way that we minimize the movement of data and guarantee their availability. In this kind of environments, however, it may be difficult to gather data accurate or relevant enough of the nodes contributing to it. Additionally, the choice of nodes should be done considerably fast in order to ensure that there is always a minimum number of replicas. Because of these restrictions, most available mechanisms of node selection are not applicable, as they usually require either large execution times or accurate information − or both − (Cabrera, Juan, Marquès, and J. Proskurnia 2014). Therefore, we have developed a method capable of providing quality results with not much information and in a fast way.

In this paper we provide a real case scenario in which this mechanism may be used, in order to improve resource allocation over random choice. This case consists in a decentralized micro-blogging application, i.e., an application where blogging is done with severe space or size constraints, typically by posting frequent brief messages about personal activities. In this micro-blogging application data is distributed

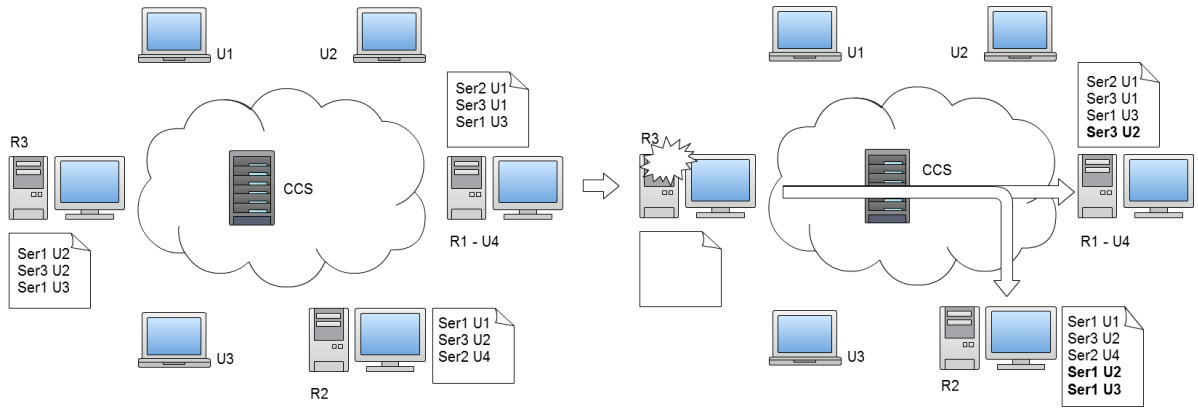

Figure 1: Structure of the micro-blogging application and process followed when a node disconnects.

among computers provided by those users who wish to collaborate the system by providing storage in their computers − usually a fraction of the total number of users; it is an example of *Distributed Computing*, a kind of *Volunteer Computing*. As information of each user is stored across different nodes, we have a centralized control system in order to route them to their information. This system is also in charge of detecting which nodes are available at each moment, deciding where to store each piece of information, and trying to guarantee that none of them is left unattended, by replicating the content of disconnecting nodes into new ones. This choice must be done as quick as possible, providing a quality option, although not necessary the optimal one −even if it is a wrong one, assignations *node-information* are temporary and will be eventually corrected. A brief scheme of the structure of the system, as well as the process it follows after a node disconnects, is provided in Figure 1.

In order to address this issue, we propose a heuristic method based on the selection of allocation nodes by a weighted function and biased randomization. The method we propose provides quality solutions in a very fast way, since it does not require costly computations in runtime. A simulation environment has been developed for a micro-blogging application, so that this selection method has been tested with the aim of setting the weights of the function that provides the suitability or quality of a node. Our final results have been compared with the results obtained by a random selection of nodes, in order to check the quality of our approach.

Finally, it is important to notice that the correct management of resource allocation is ultimately reflected in the quality of service (QoS) received by each user on the system, so it is a very important aspect to consider. In the literature, high sophisticated methods have been proposed to solve similar problems, such as the resource allocation with QoS in wireless networks (Julian, Chiang, O'Neill, and Boyd 2002), or the Reliability Redundancy Allocation Problem (Kuo and Rajendra Prasad 2000). These methods range from Genetic Algorithms (Coit and Smith 1996), Simulated Annealing (Ali, Kim, Siegel, and Maciejewski 2008), Ant Colony Optimization (Liang and Smith 2004), Particle Swarm Optimization (Yeh 2009), Iterated Local Search (Cabrera, Juan, Marquès, and J. Proskurnia 2014) to Tabu Search (Kulturel-Konak, Smith, and Coit 2003). However, these techniques are time consuming and do not match the needs of the volunteer computer network explained in this paper. They are thought to work in networks without the dynamicity that volunteer computing presents. Additionally, these methods are not thought to be applied with data volumes, but usually with services. When data replication is needed the reallocation of replicas involves an overloading in the network that is not appropriate. To the best of our knowledge, the particular problem dealt in this paper has not been tackled before in the literature.

The remainder of this paper is structured as follows. Section 2 is devoted to provide a more in depth description of both the problem and the solution proposed. Then, in section 3 we describe the computational experiments and the results we obtained. Finally, in section 4 we explain the conclusions we have extracted and the further work it could be done regarding this problem.

2 OUR APPROACH

2.1 Data Allocation

In this paper we propose a simple and fast heuristic to allocate resources to *nodes*, as well as a way of simulating the behaviour of these nodes. We will use the latter to test the goodness of the first one. Although some of the current methods of allocating resources are likely to produce better results than the ones from our mechanism, they require much computational time, and usually a considerable amount of information from the nodes (Cabrera, Juan, Marquès, and J. Proskurnia 2014). In some systems these resources may not be available and, therefore, obtaining a reasonably quality solution may be enough. Therefore, this is the motivation behind our proposed optimization method.

Our approach is based on a node-quality function that consist of parameters and weights associated to these parameters. The node quality is a property that arises and is necessary in this particular system because it is a community network where nodes give their resources voluntary. Consequently, nodes are free to disconnect and connect when they want, in contrast to dedicated servers. For this reason, we need a measure that gives us an idea of their goodness and robustness as repository.

First, for each node we collect a set of m numeric parameters $\mathscr{P} = \{p_1, ..., p_m\}$, usually those we judge most relevant, or simply those that are available. For each of these parameters we have an assigned weight: $\mathscr{W} = \{w_1, ..., w_m\}, \forall j\ 0 < w_j < 1$. These weights indicate the importance of that parameter: the higher it is the more important in the node-quality function. Ideally, the sum of these weights should amount to 1, rather to make it human friendly than for computational reasons. In order to determine which parameters are desirable and which are not, we introduce a new factor for each parameter $\mathscr{C} = \{c_1, ..., c_m\}, \forall j\ c_j \in \{-1, 1\}$. In our case we have manually chosen them according to common sense, as only 3 parameters are used and they were straightforward. However, in case more parameters were used, or they were more abstract, these factors could be learnt in the same way as the weights $\mathscr{W}$. Using all this we can obtain the quality of a node n by means of equation:

$$quality(n) = \sum_{j=0}^{m} c_j \times w_j \times p_j \tag{1}$$

This way, those parameters for which we want high values will have $c = 1$ and, therefore, will increase the overall suitability. On the other hand, penalizing parameters will have $c = -1$, and will decrease the overall quality.

In the case of our micro-blogging application we used three parameters in order to obtain the quality of each node. These parameters were chosen due to their availability and expressiveness, and others could be added in the future:

- *Percentage of connected time:* This parameter indicates the percentage of time the node has been connected (serving) since its first connection. A node with a high percentage of connected time is desirable, as the information stored in it is less likely to need to be reallocated. Its factor c will be, therefore, 1, as we want it to be as high as possible. As a possible future improvement, we could take, instead of all time since its first connection, a certain window of time to monitor.
- *Number of disconnections:* This parameter is thought to complement the previous one, and it is simply the number of times the node has disconnected. We want it to be as low as possible, and, therefore, its factor will be $c = -1$. In order to prevent the difference between magnitudes, this value is normalized between 0 and 1, taking as upper bound the highest value obtained from a repository so far. As an example of how this parameter is combined with the previous one, let's suppose node A has stayed connected for 5h since its first connection and, afterwards, it has turned off for another 5h. Node B, on the other hand, has had 5 short connections of 1h each, with 1h between each of them. The *percentage of connected time* of both of them will be 0.5, but the first node will be better, as its services will need less reallocations. Consequently, the *number of*

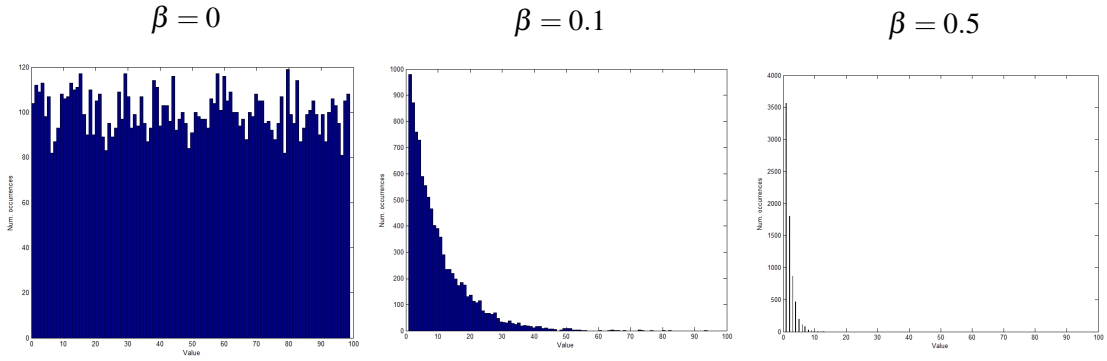

Figure 2: Influence of β parameter in the resulting distribution.

disconnections will be 1 and 5 respectively − this value is not normalized here for clarity purposes − and, therefore, the first repository will have a higher quality.

- *Percentage of occupation:* This last parameter indicates the degree of occupation of that node. It is used to prevent the saturation of nodes. Combined with the Biased Randomization explained below, it guarantees that the most capable nodes (those that stay connected most time) are not taken too fast. This allows, theoretically, for all information to have access to similar resources independently of the moment it requests them. As we want those nodes with less occupation to be selected first, we assign its factor c to -1.

By means of the quality function, we can sort a list of nodes in descending order of quality. Each time a node is required in order to allocate some resources, it is chosen from this list, by means of a Biased Randomization (Juan, Faulin, Ruiz, Barrios, Gilibert, and Vilajosana 2009, Juan, Faulin, Ruiz, Barrios, and Caballe 2010). We have chosen a geometric distribution, so that we need a parameter, β. Figure 2 shows how the β parameter influences in the geometric distribution:

As it can be seen, the higher the β, the more often the first values are chosen. By using this property, we are able to choose, from previous list, a subset of nodes in which we give priority to the best ones. This way, we ensure that good quality nodes are chosen and, at the same time, we prevent the best ones to be completely saturated too fast.

A final consideration is done regarding the order of the list. It is important to take into account that this quality is not necessarily static, and it can change over time. If this is the case, the list will need to be periodically updated to take new qualities into account. Keeping it completely up to date may require more resources than available. However, we may keep an approximate order, which will provide results close to the "true" ones, while keeping the computational charge low. In order to do so, whenever a node changes in quality and should be moved, we keep it where it is, instead of moving it to its actual position − which would take $\mathcal{O}(n)$ time. To keep an approximate order, we periodically sort the beginning of the list − in our case, we have used its first 25%−, which corresponds to the most commonly selected nodes. This sorting is done by removing all nodes there and inserting them at their correct position, and it is usually enough to deal with most of the disorder, while keeping the computational load reasonably low. At the same time, we also include a global sorting that periodically sorts the entire list, putting each node at its correct position. Even though this is necessary in order to keep the less quality nodes sorted, it is done much less often than the partial sorting. Therefore, we can have an approximate order without consuming too many resources. The intuition behind the partial sorting is that the last nodes will not change quality as often as the others. This is one particularity of the example case, as there are two situations that involve a change in the quality of a repository. The first one happens when a repository is assigned to a user, since the occupancy level affects the quality of the repository in order to avoid its saturation. The second one

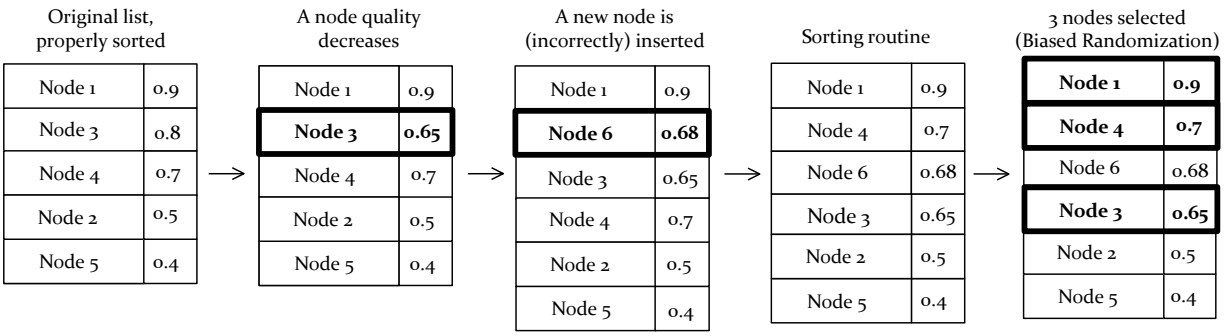

Figure 3: Process of how to keep the list pseudo-ordered, and how Biased Randomization works.

happens when the repository has been disconnected and it is connected again. From these we deduce that quality of best nodes will change more often as the others, as they will have more services assigned.

Interestingly enough, the experiments performed have suggested that, in some cases, as time passes the *approximate* list becomes quite close to the *optimal* one. An example of the usage of the list and the global sorting subroutine is shown in Figure 3, in which it can also be seen a possible selection of nodes done by *Biased Randomization*.

2.2 Weights Selection And Environment Simulation

Even though this method is expected to obtain good results, it needs some previous tuning. Concretely, the weight ($\mathscr{W}$) associated to each parameter. These weights, ranging between 0 and 1, are used to indicate the importance of the corresponding parameter: the higher they are in absolute value, the more important the parameter. These weights can be manually set, if thorough knowledge of the system is available, but they can also be "learnt" in order to find the best available combination. Since our real application is a dynamic system we have decided to use simulation with the aim of finding the combination of weights that allows the best performance of it.

This way, our proposal consists in creating a simulation of our system in which to test different weights, and choose the best combination. An additional advantage of this method is that it allows us to simulate different environments and obtain the most appropriate weight combination for each of them. Therefore, in case the real environment is modified, it is possible to quickly adapt to this change. This could be done by choosing the set of weights corresponding to the previously simulated environment that was closest to this new environment.

This simulation should be as close as possible to the real execution. In order to do so, it is important to try to reproduce the behaviour of the environment in the most realistic way. One possible way of simulating the temporal continuity of a real environment consists of replacing it by routines being run regularly, which trigger certain events.

Therefore, the solution we have used to perform an experiment starts by assigning a certain execution time. During that time, the programmed routines are triggered, all specified elements are simulated, and the required information is retrieved. This way, we obtain an approximation of how the real environment would behave. It is important to take into account that, usually, the more time a simulation is run, the more reliable its results are. Therefore, we have tried to maximize it as much as our computational resources have allowed us.

We have defined three main periods in our simulation: *initialization*, *stabilization*, and *monitoring*. In the first one, both repositories and users are gradually created, and the information of the latter is stored in the currently available repositories. At the same time, the behaviour of repositories is being simulated, they are turned on and off and their information is reallocated. The second period consists in giving some time to the system in which neither users nor repositories are added, so that it can consolidate. After this

period, the system is supposed to have reached an stable situation, representing that the initial irregular phase has already finished. Therefore, we start monitoring the system. This monitoring time corresponds to one month in real time.

In order to provide a higher degree of reliability to the results, for each configuration tested, we have run 30 times each experiment. The extracted results for each experiment could include some statistic measures of all executions. This way we can obtain the average behaviour, and reduce the possibilities of an outlier occurring.

To sum it up, in a simulation we would need:

- **Elements to simulate:** we simulated the storage nodes behaviour, by switching them on and off with different frequencies, to produce different quality nodes. We also inserted some information traffic into it to simulate the users' activity. By using this data we were to determine the quality of the simulated environment, and evaluate the weight combinations. In this part we would also need to specify how often each routine is triggered.
- **Environment settings:** as already explained, our approach allowed us to find the best weight combination for different environments, so that later we can use these combinations in anticipation to multiple scenarios. Therefore, for each experiment, or weight combination, we need to determine the features of the environment in which to be tested.
- **Goodness estimator:** in order to compare the performance of each experiment, we need a way to measure their goodness. This is a critical part of the process as, if the right parameters are not chosen, a good weight combination may pass unnoticed, whereas a bad one may, by chance, obtain a good rating. Ideally, we should have only a handful of highly relevant indicators. In our case we have defined three indicators regarding the number of *reallocations*. These indicators represent how many times a certain kind of service has been replicated, from one node to another, as a result of the disconnection of a replica. Notice that the information is replicated among different nodes in order to preserve its integrity in case a node fails, so that if this happens, the information can be replicated into a new node from some of the other ones storing it. If the selection of nodes to allocate resources is made in a wrong way, the number of reallocations will be high, since low quality nodes will be selected and due to their disconnections reallocations will be needed more frequently. Accordingly, we need this indicator to be as low as possible. There are three kinds of service in our system (*Service 1*, *Service 2* and *Service 3*) that manage different information of each user. Therefore, we have three indicators, one for the number of reallocations of each of them.
- **Experiment settings:** finally, we need to determine the weight combinations to test, as well as the number of executions we will perform for each experiment.

3 COMPUTATIONAL EXPERIMENTS

This section is dedicated to assess the performance of our heuristic method inside the simulation environment proposed in this paper. This approach has been implemented using the programming language Java Standard Edition 7.0. All the computational experiments have been carried out on a PC equipped with CentOS 6.6, an AMD processor with 4 CPU cores, 2.3GHz with 4GB of RAM.

In our experiments we have considered an scenario of the micro-blogging application, with three kinds of nodes (depending on their quality): *low*, *middle* and *high*. Each of these kinds of nodes behaves differently, and allows us to simulate different scenarios, from *difficult* ones, with lots of *low* quality nodes, to *easier* ones, in which prevail the *high* quality type. To test our experiments we have decided to take a reasonably "pessimistic" approach. Thus, we have an scenario with half of the nodes being of *low* quality, since we assume it is the most typical real scenario.

In these experiments we have defined that services 1 and 3 must have their information replicated 3 times. This value has been selected in an intuitive way, and we plan to perform further experiments to determine if there is a better value. On the other hand, service 2 has its information stored in a single

Table 1: Settings of the experiment.

Parameter	**Value**
Users	700
Nodes	100
Low quality	50
Middle quality	30
High quality	20
Executions/experiment	30
β	0.25
Minimum n. of services	200
Maximum n. of services	220

replica. The reason is that this service is the combination of the service 1 of some different users, so it can be rebuilt from those, although with a certain additional cost. However, keeping it updated among different repositories proved troublesome, so we have considered that rebuilding it is the best option.

The exact settings we have used, to define the environment and the experiment itself, are provided in Table 1:

- *Users*: The number of users (and, therefore, services) being simulated.
- *Nodes*: The total number of nodes being simulated, as well as its breakdown into each different category.
- *Executions/experiment*: How many times each experiment has been run.
- β: The β parameter of the Biased Randomization.
- *Minimum and maximum n° of services*: The capacity of each node is randomly generated between these two numbers.

In Table 2 we provide the results of the experiments we have performed. The first column corresponds to the identification of the experiment, and the rest of columns are divided between the weights being tested and the results obtained. For the weights, column "Connection" corresponds to the percentage of connected time parameter, columns "Occupation" corresponds to the percentage of occupation parameter, and column "Disconnections" corresponds to the number of disconnections parameter. Notice that the first row, where weights are not specified, corresponds to the random performance of the systems, i.e., without using the quality to sort the nodes and choosing them randomly.

In the results section, each column corresponds to the number of times one of such services has been replicated. Therefore, the highest the value, the greatest the load on the system. Notice that, as Service 2 is replicated only once, it has the lowest value. We have highlighted the best two results for each kind of service. The experiment number 12 has obtained one of the two best results at each of the three categories, being the best one in two of them, and only 0.3 away from the best result in the remaining one. From these results, we can say that, in the scenario that we have checked where half of the nodes are of *low* quality, the most important parameter is the percentage of connected time, with a weight value 0.8. The number of disconnections has a marginal importance in this case with a weight value 0.05, and the percentage of occupation has certain relevance with a weight value 0.15.

Table 3 depicts a comparison between the base case − in which we use the random choice − and the best result we have obtained. For the three kinds of service the relation between the base results and our best results is constant. This fact was expected, but it gives, nevertheless, more reliability to the results. Regarding the review of the results, we have obtained an improvement of 33% with respect to the random choice by introducing only a negligible overload to the system due to the simplicity of our methodology.

Table 2: Results of the experiments.

ID	**Weights**			**Results**		
	Connection	Occupation	Disconnections	Service 1	Service 2	Service 3
0	-	-	-	15631.97	5150.27	15609.87
1	0.33	0.33	0.33	10731.17	3518.23	10689.83
2	0.6	0.1	0.3	10472.6	**3426.57**	**10485.6**
3	0.6	0.3	0.1	10557.4	3480.43	10575.4
4	0.3	0.1	0.6	**10470.07**	3433.73	10502.07
5	0.1	0.3	0.6	10542.87	3448.8	10514.07
6	0.1	0.6	0.3	10821.43	3557.97	10814.47
7	0.3	0.6	0.1	10871.23	3595.7	10892.67
8	0.6	0.2	0.2	10770.53	3543.03	10731.53
9	0.2	0.2	0.6	10516.6	3454.67	10489.77
10	0.2	0.6	0.2	10685.03	3511	10672.43
11	0.8	0.05	0.15	10577.7	3487.7	10562.03
12	0.8	0.15	0.05	**10470.37**	**3425.17**	**10413.17**
13	0.15	0.05	0.8	10665.2	3522.17	10678.53
14	0.15	0.8	0.05	10814.23	3557.2	10798.43
15	0.05	0.15	0.8	10670.63	3539.47	10715.97
16	0.05	0.8	0.15	10934.33	3589.43	10981.27
17	0.8	0.1	0.1	10673.67	3545.07	10708.57
18	0.1	0.1	0.8	10970.73	3597.3	10949.5
19	0.1	0.8	0.1	11150.47	3660.7	11196.07

Table 3: Comparison between base − random − and our best results.

	Base results	**Best results**	$\frac{Best-Base}{Base} \times 100$
Service 1	15631.97	10470.07	−33.03
Service 2	5150.27	3425.17	−33.50
Service 3	15609.87	10413.17	−33.29

4 CONCLUSIONS

In the context of Volunteer Computing, we have considered a real micro-blogging application. In order to provide a quality behaviour of this application it is necessary to properly manage the resource allocation, since the reliability of the nodes of this distributed system may be low. The procedure to insert in this system for managing the resource allocation cannot be time consuming and has to involve improvements regarding a random performance.

In contrast to sophisticated methods proposed in the literature, we have proposed a simple heuristic method capable of obtaining quality results in an extremely fast way. This heuristic uses a weight system to quickly determine each resource quality and a biased random procedure to select them accordingly. In order to tune and test it, we have prepared a simulation environment for the real micro-blogging application, so that we can obtain reliable results that allow us to deliver the optimization system to a real environment.

This way, we have tested the proposed method in a simulation of a micro-blogging decentralized application, in order to check the usefulness of the former. By doing so, we have been able to decrease the number of reallocations in the system by 33% with respect to using a random selection, without introducing a noticeable load in the system − that is, after the training period.

As further work, we would like to compare the performance of our heuristic using the *Biased Randomization* with both a greedy approach choosing always the first elements of the list or a more sophisticated *GRASP* (Feo and Resende 1995) algorithms, in order to see which one of them provides the best results. Additionally, various scenarios can also be tested, as well as different parameters combinations −such as β. Regarding the simulation of the environment, as a possible improvement, we have considered the possibility of allowing reconnecting nodes to keep serving some of the services they had before disconnecting, e.g., in case they are reasonably up to date. This way, we can save having to completely replicate some of the services.

ACKNOWLEDGMENTS

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness (TRA2013-48180-C3-P and TRA2015-71883-REDT), and FEDER.

REFERENCES

Ali, S., J.-K. Kim, H. J. Siegel, and A. A. Maciejewski. 2008. "Static heuristics for robust resource allocation of continuously executing applications". *Journal of Parallel and Distributed Computing* 68 (8): 1070 – 1080.

Anderson., D. 2006. "Boinc: A system for public-resource computing and storage". In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE Inter-national Symposium*, Volume 1, 73–80. IEEE.

Anderson, D., and G. Fedak. 2004. "The computational and storage potential of volunteer computing". In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop*, 4–10. IEEE.

Cabrera, G., A. L. Juan, D. Marquès, and I. J. Proskurnia. 2014. "A simulation-optimization approach to deploy Internet services in large-scale systems with user-provided resources". *Simulation* 90 (6): 644–659.

Coit, D. W., and A. E. Smith. 1996. "Reliability optimization of series-parallel systems using a genetic algorithm". *IEEE Transactions on Reliability* 45 (2): 254–260, 263. cited By 397.

Feo, T. A., and M. G. C. Resende. 1995. "Greedy randomized adaptive search procedures". *Journal of Global Optimization* 6 (2): 109–133.

Juan, A., J. Faulin, R. Ruiz, B. Barrios, and S. Caballe. 2010. "The SR-GCWS hybrid algorithm for solving the capacitated vehicle routing problem". In *Applied Soft Computing*, Volume 10, 215–224.

Juan, A., J. Faulin, R. Ruiz, B. Barrios, M. Gilibert, and X. Vilajosana. 2009. "Using oriented random search to provide a set of alternative solutions to the capacitated vehicle routing problem". In *Operations Research and Cyber-Infrastructure*, Volume 47, 331–346.

Julian, D., M. Chiang, D. O'Neill, and S. Boyd. 2002. "QoS and fairness constrained convex optimization of resource allocation for wireless cellular and ad hoc networks". Volume 2, 477–486. cited By 89.

Kulturel-Konak, S., A. Smith, and D. Coit. 2003. "Efficiently solving the redundancy allocation problem using tabu search". *IIE Transactions (Institute of Industrial Engineers)* 35 (6): 515–526. cited By 124.

Kuo, W., and V. Rajendra Prasad. 2000. "An annotated overview of system-reliability optimization". *IEEE Transactions on Reliability* 49 (2): 176–187. cited By 315.

Liang, Y.-C., and A. Smith. 2004. "An ant colony optimization algorithm for the redundancy allocation problem (RAP)". *IEEE Transactions on Reliability* 53 (3): 417–423. cited By 166.

Yeh, W.-C. 2009. "A two-stage discrete particle swarm optimization for the problem of multiple multi-level redundancy allocation in series systems". *Expert Systems with Applications* 36 (5): 9192–9200.

AUTHOR BIOGRAPHIES

XAVIER SERRA is an Associate Researcher at Computer Sciences, Multimedia & Telecommunication Studies at Universitat Oberta de Catalunya (UOC) since 2014. He graduated as Computer Science Engineer from the Facultat d'Informtica de Barcelona (UPC) in 2015, and is currently taking an interuniversity Master in Artificial Intelligence from UPC, UB and URV. His current research is based in heuristics and machine learning. His email address is xserralza@gmail.com.

Dr. JESICA DE ARMAS obtained her PhD in 2012 from Universidad de La Laguna working on optimization algorithm for logistics problems. Then, she was contracted as postdoc by the same university. During this period she developed several optimization algorithms for different transportation problems. Additionally, she established technological transference with some companies. Nowadays, she is working as postdoc researcher at Computer Sciences, Multimedia & Telecommunication Studies at Universitat Oberta de Catalunya (UOC). Her current research interests include high performance computing, metaheuristics, and optimization, mainly in the logistics and transportation areas. Her email address is jde_armasa@uoc.edu.

Dr. JOAN MANUEL MARQUÈS is an Associate Professor at Computer Sciences, Multimedia & Telecommunication Studies at Universitat Oberta de Catalunya (UOC) since 1997. He graduated as a Computer Science Engineer from the Facultat d'Informtica de Barcelona (UPC) in 1991 and received his PhD from UPC in 2003. His research interests include the design of scalable and cooperative Internet services and applications. He is member of the Association for Computing Machinery. His email address is jmarquesp@uoc.edu.