Multi Criteria Biased Randomized Method for Resource Allocation in Distributed Systems: Application in a Volunteer Computing System

Javier Panadero[a], Jesica de Armas[b], Xavier Serra[a], Joan Manuel Marquès[a]

[a] *IN3 - Computer Science Dept., Open University of Catalonia, Barcelona, Spain*
{jpanaderom, jmarquesp}@uoc.edu, xserralza@gmail.com
[b] *Department of Economics and Business, Universitat Pompeu Fabra, Barcelona, Spain*
jesica.dearmas@upf.edu

Abstract

Volunteer computing is a type of distributed computing in which a part or all the resources (processing power and storage) necessary to run the system are donated by users. In other words, participants contribute their idle computing resources to help running the system. Due to the fact that the nodes which compose the system are provided by a large number of users instead of a single (or a few) institution, a main drawback of volunteer computing is the unreliability of these nodes. For this reason, the selection of nodes to be involved in each task becomes a key issue. In this paper, we propose the Multi Criteria Biased Randomized (MCBR) method, a novel selection method for large-scale systems that use unreliable nodes. MCBR method is based on a multicriteria optimization strategy. We evaluated the method in a microblogging social network formed by a large number of microservices hosted in nodes voluntarily contributed by their participants. Simulation results show that our proposal is able to select nodes in a fast and efficient manner while requiring low computational power.

Keywords: Distributed Computing, Volunteer Systems, User assignment, Allocation methods, Resource provisioning

1. Introduction

Volunteer Computing (VC) [1] systems are large-scale heterogeneous distributed systems where resources (nodes) are donated by volunteers. Public contributors share a part of their idle computational resources to execute computationally expensive applications.

This kind of computation has become increasingly popular due to the fact that it provides a scalable, elastic, practical, and low cost platform to increase the computational and storage demands of many applications. However, the nodes are provided by users in a voluntary way, which means that they may suffer from a lack of reliability, since they are usually non-dedicated and dynamic. Therefore, the system must be able to tolerate both sudden connections and disconnections of nodes. An efficient mechanism to select which nodes will run a job or store some data is of high importance for two main reasons: (a) it is necessary to guarantee the fulfillment of the task or the availability of the data; and (b) it is recommended to minimize the quantity of nodes required for it.

Regarding to this second aspect, it is important to minimize the number of replicas involved to provide the service, specially from the storage point of view: each time a node fails a new node must be selected and all data must be replicated into it. In a VC environment with not enough highly available nodes, the selection mechanism should be able to combine nodes with different availability levels to guarantee that the system provides a good quality of service (QoS). In addition, this mechanism should be fast in order to quickly react to changes in the system.

In this paper we propose the *Multi Criteria Biased Randomized* (MCBR) method, a novel selection strategy for large-scale systems composed of unreliable nodes. MCBR allows to select the most suitable nodes in an efficient and fast way, ensuring a minimum QoS to the users. The proposed method is based on ideas of the Lexicographic Ordering (LO) multicriteria optimization strategy [2]. Thus, MCBR is a hierarchical method in which the intrinsic properties of the

nodes are categorized according to different priority levels. Then, a sequence of decisions is made following the previous established priority order. Biased randomization techniques [3] are used to distribute and balance the load of the nodes. The proposed method provides high quality solutions in a very fast way, since it does not require costly computations in runtime. Moreover, due to the flexibility of this method to prioritize the properties of the nodes, it can be applied to a wide range of large-scale distributed systems other than VC, as could be P2P or Grid Systems.

We tested MCBR by simulating a real large-scale social network called Garlanet [4], that stores all data in computers voluntarily contributed by its participants. More precisely, for each user, Garlanet deploys a set of replicated microservices (in the voluntarily contributed nodes) that are in charge of guaranteeing the availability of the data. The MCBR method is used to select which node will allocate each replica of each microservice.

To validate and quantify the quality of MCBR, we have developed a metaheuristic [5]. Metaheuristic algorithms are widely recognized as efficient approaches for many optimization problems. They focus on exploring the search space to obtain optimal or quasi-optimal solutions in a reasonably short time. The metaheuristic developed in this work allows us to compare the evolution experimented by the system when applying MCBR in real time, with the results of a near optimal selection of nodes obtained with it. The experimental validation proves that the MCBR method provides high quality solutions, ensuring the minimum QoS and avoiding the excess of data movement. This last point is crucial when selecting a solving method for this kind of systems.

The remainder of this paper is structured as follows: Section 2 presents a literature review on similar approaches. Section 3 is devoted to describe the proposed MCBR method. Then, Section 4 presents the prediction quality model needed by the MCBR method. In Section 5, the metaheuristic used to compare our results is described. Section 6 presents a complete set of experiments and analyses the results. Finally, Section 7 concludes this work and proposes possible future research lines.

2. Related Work

Several recent works in the literature have focus their attention on the selection of resources in distributed large-scale systems based on heterogeneous and non-dedicated components, due to the importance of making an efficient use of the resources in them. Thus, next sections are devoted to go through the main works about it. As mentioned in each section, none of these works solves the particular problem at hand.

2.1. Resource allocation in VC systems

Since the efficient resource allocation is a key factor in VC systems, several authors have worked on this research line. Estrada et. al [6] propose a distributed evolutionary genetic algorithm to design scheduling policies in VC, which maximize the throughput of the system. The proposed algorithm automatically generates scheduling policies that increase throughput across a variety of different VC projects, in contrast to the manually-designed policies, which are limited to increasing throughput for single projects. The algorithm is based on searching over a wide space of possible scheduling policies, using a small subset of IF-THEN-ELSE rules, which are used to generate the most suitable policies.

Ghafarian et al [7] [8] focus on proposing a method to schedule scientific and data intensive workflows, to enhance the utilization of VC systems. The proposed method increases the percentage of workflows that meet the deadline, satisfying the QoS constraints in terms of the deadline, minimum CPU speed, and minimum RAM or hard disk requirements. The proposed workflow scheduling system partitions a workflow into sub-workflows, to minimize data dependencies among the sub-workflows.

Sebastio et. al [9] propose a distributed framework to allocate tasks in large-scale Volunteer Clouds platforms, according to different scheduling policies. The framework takes into account five different policies, which attempt to maximize the number of executed tasks and minimizing the time at which the execution ends, both for the entire task set and for each task in the set.

Each policy is formalized as a mathematical optimization problem with constraints, which is solved in a distributed fashion. In order to solve the problem in a distributed way, the framework uses the Alternating Direction Method of Multipliers (ADMM) [10] to decompose the optimization problem. Then, it is distributed and independently solved by the volunteer nodes. Besides the throughput, another important point to consider by users of Volunteer Clouds platforms is the money budget. Guler et al [11] propose various heuristics to distribute jobs, while maximizing the throughput done by the users, without violating established money budget constraints. The heuristics are based on the price of electricity consumed by the peers, considering its temporal variation during the time, and the CPU time used.

These previous approaches are focused on maximizing the throughput of the VC system under some constraints, taking into account the types of jobs/tasks to execute in the system previously. Unlike these works, our method is focus on the resources selection in dynamical real time environments, trying to quickly react to changes in the system, e.g., sudden disconnections or the arrival of new users to the system.

2.2. Resource allocation in Distributed Social Networks and Applications

Due to the increasing popularity of social networks, other works have focused on the assignment of resources in Online Distributed Social Networks, which run over large-scale distributed systems. Thuan et al [12] propose three heuristic algorithms for solving the client-server assignment problem in online social network applications. The algorithms are based on the user communication patterns. The authors objective is to find an approximately optimal client-server assignment that results in small total communication load, while maintaining a certain level of load balance.

Zhang et al [13] propose three heuristics to assign clients to servers in continuous Distributed Interactive Applications (DIA) [14]. The heuristics are focused on reducing the network latency for maximizing the interactivity under consistency and fairness requirements. They are based on analyzing the minimum

achievable interaction time for DIA's to preserve consistency and provide fairness among clients. Zheng et al [15] add a complementary study to the previous work. Authors present two efficient server placement algorithms for hosting continuous DIA's. These algorithms are addressed to find optimum locations of servers in the network, with the goal of optimizing the interactivity performance, while maintaining the consistency and fairness of DIA's. The proposed algorithms take into account the interaction between clients, considering their path in the network and the latency, to produce near-optimal server placements.

Hiroshi et al [16] present a heuristic algorithm via relaxed convex optimization, that takes a given communication pattern among the clients, providing an approximately optimal client-server assignment for a pre-specified trade-off between load balance and communication. This heuristic can be used in distributed applications such as Instant Messaging Systems (IMS).

The proposed methods in these works are based on profiling the user behaviors (i.e. obtaining information about the user communication patterns), to find optimal client-server assignments in large-scale distributed systems. However, the MCBR method does not need to gather user behavior to make optimal assignments. All the information needed is obtained from the nodes that compose the distributed system.

2.3. Resource allocation in Cloud Computing Systems

In a more general context, with the advent of the cloud and federated clouds [17], Coutinho et al [18] proposed the Cloud Resource Management Problem (CRMP). The CRMP is a multi-criteria optimization problem which consists of assigning resources to users, taking into account both cost and performance preferences of consumers for supporting purchasing. To solve the problem, an Integer Programming (IP) formulation, and a GRASP heuristic [19], called GraspCC, are presented by the authors. Both methods consider time and budgets limits of consumers, and different application requirements in terms of resource demands. Authors claim the need for both approached, since exact procedures have often proved incapable of finding optimal solutions in real-

world problems, as they are extremely time-consuming. Conversely, heuristics and metaheuristics provide sub-optimal solutions in a reasonable short time.

More recently, the same authors have published a new work [20] addressing the CRMP in multi-cloud environments. They propose GraspCC-fed, a GRASP heuristic approach for dimensioning the amount of virtual machines to allocate for a parallel workflow in federated cloud environments, before its execution. GraspCC-fed takes into account both costs and execution times in a weighted sum objective function. Same problem has also been resolved by Heilig et al [21] using a Biased Random-Key Genetic Algorithm (BRKGA) [22]. They propose the BRKGA-MC, which is based on a cloud brokerage mechanism. The BRKGA-MC is a deterministic algorithm that takes as input a vector of n random keys and it returns a feasible solution of the optimization problem at hand along with its objective value. The algorithm is able to determine a feasible solution in the millisecond range with an excellent quality, and it is suitable for being included as a real-time decision support tool in related deployment processes.

As explained in the next section, due to the flexibility of the MCBR method to adapt the parameters and the use of priority levels, it could be applied to a wide range of large-scale distributed systems. Although it is not the purpose of the present work, it could be used to solve this multi-objective problem, maintaining the fast and efficient node selection process.

3. Multi Criteria Biased Randomized Method

This section presents the MCBR method, which focuses on selecting the most suitable nodes to allocate resources, in an efficient and fast way.

In a previous work [23], we proposed a multicriteria optimization approach based on a node-quality function. This method consists of parameters and weights associated to these parameters (weighted-sum optimization method), in which multiple objective functions are combined to form a single function.

This method is effective when using a reduced number of parameters to op-

timize. However, it becomes more challenging when the numbers of parameters increases. This is because the best values of the weight factors cannot be easily determined, since: (a) the numerical quantities are typically not based on a uniform scale; (b) the number of objective functions can be large; and (c) the consequences of a given trade-off cannot be quantitatively known prior to the optimization.

In order to overcome this issue, this paper proposes the MCBR method, which is a hierarchical allocation method based on *Lexicographic Ordering (LO)*, traditionally used in multi-objective combinatorial optimization problems. The MCBR method categorizes the parameters to optimize into different priority levels, providing good quality solutions in a fast and efficient way. Using this approach, we avoid to categorize the parameters of the objective function quantitatively. With the aim of making possible its use in a wide range of systems, the MCBR method allows an easy-to-use adaption of the parameters and their priority levels.

3.1. Hierarchical allocation method

As mentioned before, the MCBR method is based on the concept of a multi-criteria optimization strategy called *Lexicographic Ordering (LO)*. This method potentially avoids the use of weight factors by incorporating priorities of the individual planning criteria (objective functions) explicitly in the optimization process.

The LO method assumes that the objectives can be ranked in order of importance (from best to worst). The optimal value is then obtained by minimizing/maximizing the objective functions sequentially, starting with the most important one and proceeding according to the order of importance of the objectives. Thus, this multi-objective optimization technique can be represented as an objective function $F(x) = [f_1(x), f_2(x), ..., f_N(x)]$, which contains a collection (i.e., a vector) of N individualized functions ($f_i(x)$) ordered by importance, so that $f_1(x)$ is the most important and $f_N(x)$ the least important. Additionally, the optimal value found for each objective is added as a constraint for

subsequent optimizations. This way, the optimal value of the most important objectives is preserved.

Mathematically, this method can be modeled as an ordered sequence of real objective functions with a set of constraints as follows:

$$\text{Min/Max} \quad f_i(x) \tag{1}$$

subject to:

$$f_j(x) \leq f_j(^*_j) \tag{2}$$

where $i = \{1, 2, ..., N\}$ and $j = \{1, 2, ..., i-1\}$.

As the method progresses down from level 1 to level N (the last level), the preceding objective functions are converted to new constraints with boundary values f_j^*, set by the a priori attained solutions $min/max f_j(x)$, subject to the constraints from the upper level. Accordingly, the number of constraints increases with each level up to $N-1$, reducing the feasible search space gradually in each new level.

3.2. MCBR components

Figure 1 shows an overview of the MCBR method, which is composed of two main steps. During the first step, given the complete list of active nodes and a set of criteria parameters, ranked by their order of importance, an iterative procedure based on the LO method is applied to obtain a reduced set of best nodes according to each criteria. For each iteration of the procedure, the list of nodes is sorted by a criteria parameter. Afterward, we apply a Biased Randomization (BR) [3] mechanism to select a set of best nodes for this criteria parameter, discarding the remaining nodes. This subset of nodes is provided as input to the next iteration of the procedure, and the procedure is repeated for each criteria parameter. Finally, we obtain a reduced sublist of best nodes. Subsequently, during the second method stage, a Biased Randomization (BR) mechanism is applied to the final list to select the nodes to be used.

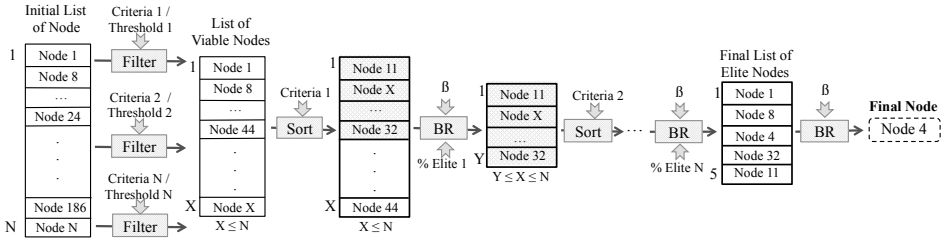


Figure 1: Overview of MCBR method

More specifically, during the first stage, the initial list of active nodes is filtering by the criteria parameters. The set of criteria parameters used by the MCBR method are represented as a vector of tuples as follows:

$$C = [\{1, threshold_1, \%elite_1, type_1\}, ..., \{N, threshold_N, \%elite_N, type_N\}] \tag{3}$$

The first parameter of the tuple represents the importance of the parameter, with 1 representing the most important. The second parameter represents the threshold value, i.e., the minimum or maximum value that can be accepted to meet the requirement of a criteria parameter. The third value represents the maximum percentage of nodes to be selected to obtain the next list of high quality nodes according to a criterion, i.e., the nodes with high values - maximizer parameter - or low values - minimizer parameter - for a criteria parameter, hereinafter 'elite nodes'. Finally, the last parameter indicates if it is a maximizer parameter (type = 1) or a minimizer parameter (type = 0).

Using this filtering procedure at the beginning of the first stage, the nodes that do not meet the minimum quality requirement imposed by the threshold value of each the criteria parameter are removed, reducing the list of active nodes. Therefore, we are trying to carry out an efficient sorting procedure, avoiding to sort nodes without possibility of being selected as candidate nodes. In case all nodes meet the requirement of this parameter, the size of the list will be the same that the size of the initial list.

Once this initial filtering procedure is done, we obtain a non-sorted list of

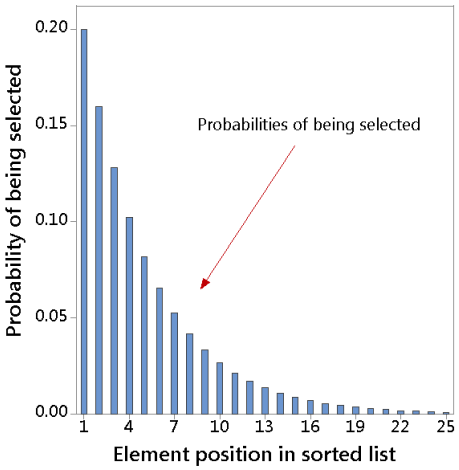
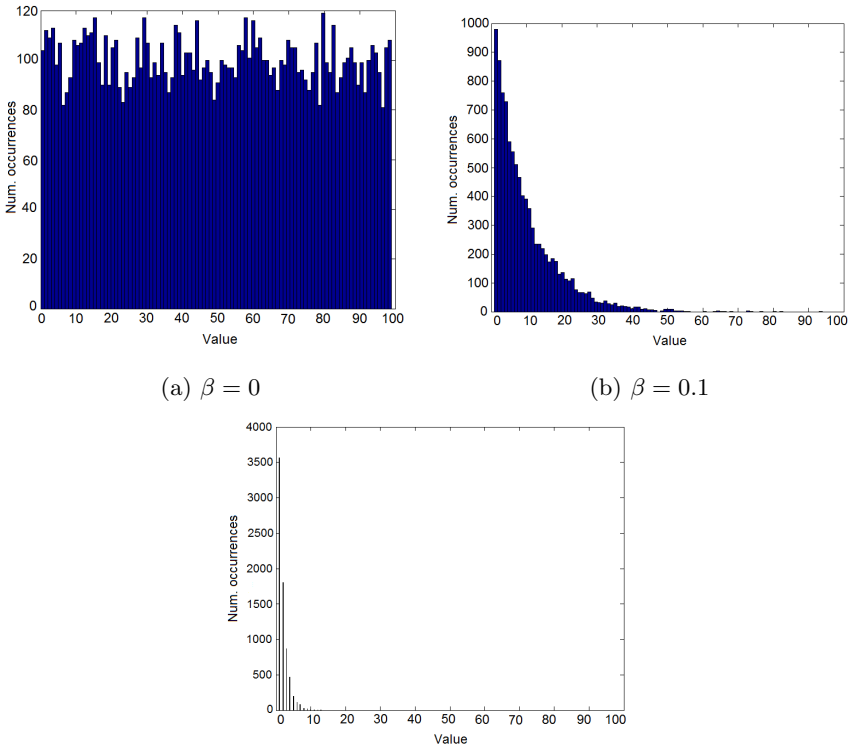


Figure 2: Biased randomization using geometric distribution with β 0.2

viable nodes, where all nodes included fulfill the requirements to be used to allocate resources. Subsequently, the MCBR method chooses the nodes to be used from this list, applying an iterative procedure which takes into account the priority of the criteria parameters.

Firstly, the list of viable nodes is sorted by the first parameter, which is the most important. Depending on whether it is a maximizer parameter or a minimizer parameter, the list will be sorted in an ascendant or descendant way. When the list is sorted, a percentage of nodes is selected until the maximum elite percentage value of the criteria parameter ($\%elite_x$) is reached. The selection of the elite nodes is carried out by means of a Biased Randomized (BR) method [3]. This method consists of using a non-uniform and non-symmetric (biased) distribution, such as the geometric distribution or the decreasing triangular distribution, instead of using the uniform distribution. In this work we have used a geometric distribution. The used of the geometric distribution associated to BR has been proven to be a good combination in the literature [24]. Figure 2 shows the probabilities of being selected in a list with 25 elements using a β parameter 0.2. The geometric distribution depends on that β parameter. Figure 3 shows how the β parameter influences the geometric distribution. As can be



(a) $\beta = 0$ (b) $\beta = 0.1$

(c) $\beta = 0.5$

Figure 3: Influence of β parameter on the resulting distribution

seen, the higher β values leads to increase the times the first values are chosen. We set this parameter to 0.3 as this value is best suited for the experimentation system used, but it can be modified as input parameter of the MCBR method.

Using a BR procedure, we ensure that nodes on the top positions of the list will be selected, i.e., the highest quality nodes for a criterion, diversifying the node selection. If we would apply a uniform selection finishing when the maximum percentage value of nodes to be selected is reached, there could be nodes on the top of the list that never would be chosen although they were quality nodes. Therefore, the BR procedure allows diversification and load balancing, keeping the logic behind the sort.

Once the elite nodes have been selected considering the first parameter, we

obtain a new reduced sublist, which keeps only a set of elite nodes for the next specific parameter. By default, we keep the 10% of the total nodes of the original list for each criteria parameter, although this value is an input parameter which can be modified. The obtained sublist will be sorted again by the next criteria parameter in the next iteration, and the elite nodes will be selected taking into consideration that criteria parameter. This procedure (sort/BR) is repeated iteratively for all the criteria parameters, until the final list of elite nodes to be used is obtained.

The algorithm used to sort the list is the QuickSort [25]. We have selected this algorithm since its average complexity is $O(n*log(n))$. Thus, the asymptotic complexity of our algorithm (lower bound) will be $O(n * log(n) * t)$, where t is the number of sorts, which depends on the number of criteria parameters used in the algorithm. Concerning to the worst case of our algorithm, it is closely related to the QuickSort algorithm, whose complexity is $O(n^2)$ in the worst case. Hence, the worst asymptotic complexity (upper bound) of our method will be $O(n^2 * t)$. Note that in each iteration of the algorithm, the n variable decreases, since we select a subset of nodes (10% of the overall).

Once the final list of elite nodes is obtained, it is used in the second stage of the method. Thus, when a node is required, it is selected from this final list. The selection of a new node is also carried out by means of using the BR method.

The objective to use BR is to assure that good quality nodes are chosen and, at the same time, preventing that the best ones to be completely saturated too fast. As in the first stage, we have used a geometric distribution, with a β parameter set to 0.4.

It is important to notice that depending on the system, the final list could be not static, and the quality of the nodes can change over time. For this reason, depending on the variability and dynamism of the system, the list should be periodically updated.

The proposed method can be used in a wide range of distributed systems such as the mentioned in the related work section. In order to validate the

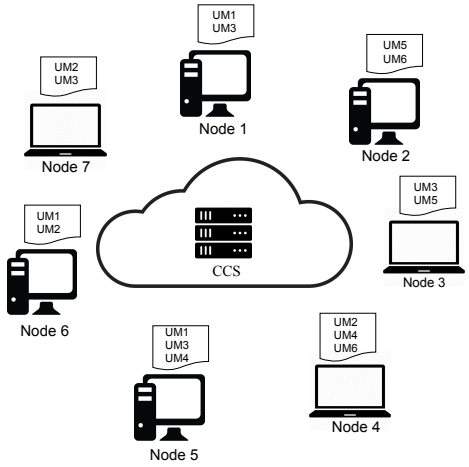
method, we have used a real case scenario of a large-scale distributed social network (Garlanet) based on VC nodes. The following section provides more details about it.

3.3. Garlanet Simulation environment

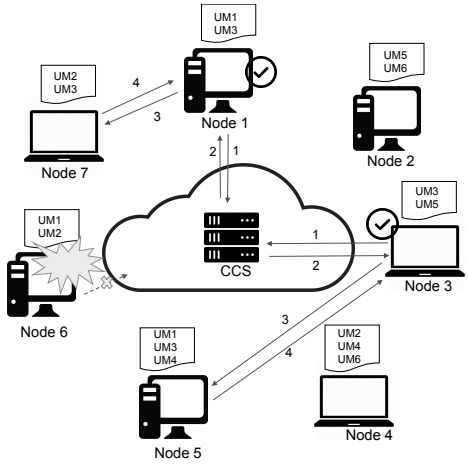
Garlanet [4] is a Twitter-like decentralized alternative implementation of a microblogging social network, that stores all its data in computers voluntarily contributed by its participants. In more detail, messages and data information of a user are handled by a microservice. Each user has her/his microservice that is replicated across different nodes to guarantee its availability. Replicas of a microservice follow the eventual consistency model. Additionally, Garlanet has a Centralized Control System (CCS) responsible for detecting available nodes at any moment and assigning the most suitable nodes to each microservice instance. Moreover, the CCS guarantees that all users have the minimum number of nodes assigned and the minimum quality of service (QoS). The quality of each user is defined as the sum of the quality of the nodes that host its data.

Garlanet imposes these two user constraints to deal with the unreliability of the nodes. The main purpose of these constraints is to try of guaranteeing all the time the availability of the user data, avoiding critical situations, where users access to the system and they have not access to their data. This way, if the available nodes do not have highly enough quality in an instant of time, and the user do not reach the minimum QoS with the minimum number of nodes, new nodes are assigned until fulfill this constraint. Using these two metrics, Garlanet tries to guarantee the data availability, reflecting in a better QoE.

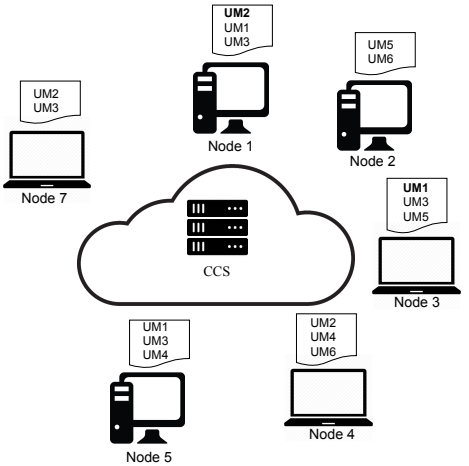
Figure 4 provides a brief scheme of the actions that occur after a node disconnects. Figure 4 (a) shows the initial state: connected nodes and User Microservices (UM) hosted in the nodes. Some time afterward, in Figure 4 (b), node 6 disconnects and the UMs kept in this node are no more available. Since nodes send heartbeat signals to the CCS, this, after some time without receiving them, will consider that node 6 is disconnected. Next, the CCS will select new nodes to replicate the UMs kept in node 6 (from users 1 and 2). In this example,



(a) Initial State



(b) Replication data process



(c) Final State

Figure 4: Replication procedure in Garlanet

the CSS decides that UM from user 1 will go to node 3, and UM from user 2 will go to node 1. Once nodes 1 and 3 are aware that they should host the UMs from users 1 and 2 (respectively), they ask for the node list in charge of hosting currently these UMs (1) to the CCS. Then, the CCS sends the node list that hosts the UMs to replicate them (2). After that, the nodes (both node 1 and node 3) select randomly a candidate node of the node list to replicate the UMs, and they start a replication session with the selected nodes (3) and (4). Finally, Figure 4 (c) shows the final situation of the system. As can be seen, the UMs of users 1 and 2 have been replicated into node 3 and node 1 respectively.

We have developed a simulator using Java Standard Edition 7.0, which tries to reproduce the behavior of the above-mentioned environment in the most realistic way. As is shown in Figure 5, the simulator is composed of three main modules: *Initialization*, *Activity* and *Control*.

The first module *(Initialization)* is responsible for initializing the environment. It creates both the users and nodes with their properties. We have considered as properties of a node: the maximum number of UM that it can host, the download speed, the probability of disconnection, the probability of reconnection and its quality. The first four are provided as input parameters and the last is predicted using the simulator. We have assumed that all nodes can host the same number of maximum UMs. The download speed of each node is randomly established between 1 MB/sec and 20 MB/sec. Regarding the users, they have as properties their minimum quality and the minimum number of replicas of their UMs (nodes to use per user).

The second module *(Activity)* is made up of two submodules: the *Activity Generator* submodule and the *Prediction Quality Model* submodule. The *Activity Generator* submodule is in charge of modifying the state of the nodes. They are turned on and off following a probability. We have defined three kinds of nodes in function of their quality: *low*, *middle* and *high*. *Low* nodes have a high probability of disconnection and a low probability of reconnection, while *High* nodes have a low probability of disconnection and high probability of connection. Each of these kinds of nodes behaves differently, and allows us to simulate

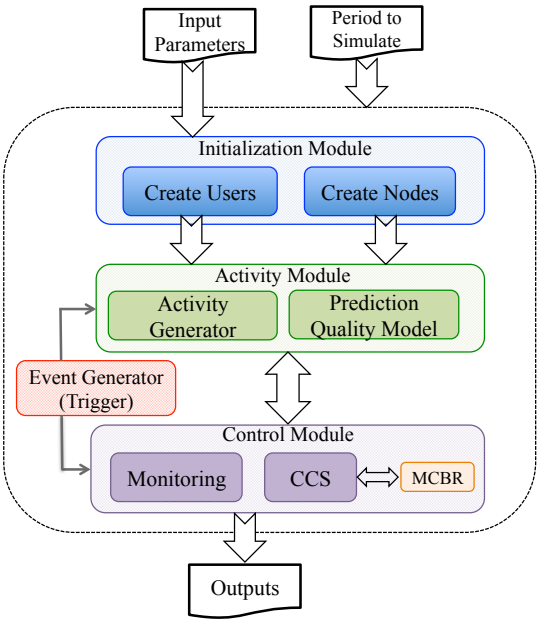


Figure 5: Simulator architecture overview

different scenarios. On the other, the *Prediction Quality Model* submodule is responsible of predicting the quality of each node.

The third module *(Control)* is composed of two submodules: the *CCS* submodule and the *Monitoring* submodule. The *CCS* submodule simulates the CCS of Garlanet. It detects the state of the nodes, selects the most suitable nodes to allocate UMs and guarantees that the users have the minimum QoS and the minimum number of nodes. As in the Garlanet real environment, these minimum QoS and number of nodes are input parameters, and their values have been assigned based on the user experience. Other systems with more dynamic behavior in function of the state could be monitored by another module, and the parameters values would be periodically updated. Notice that each time a node is assigned to a user, we consider a time lapse to simulate the data transfer time (replication session). Currently, this time is constant. After this time, the node is assigned to this user. As we can see in Figure 5, we have integrated the MCBR method with this submodule, with the objective of using this mechanism

to select the most suitable nodes. The second submodule, *Monitoring* submodule, is in charge of monitoring the system during the simulation process, and providing the output files of the simulation.

Regarding the simulation process, we simulate three months of the real system, which corresponds to 30 minutes of simulation. The total time of the real system to simulate is an input parameter, which can be set up by the user. We have defined three main periods of simulation: *Initialization*, *Stabilization*, and *Monitoring*. The first one is carried out when the simulation starts and it calls to the *Initialization* module to create users and nodes. The second period (*Stabilization*) consists in giving some time to the system to generate the prediction quality model of each node. In this period the *Activity Generator* and the *CCS* submodules respectively start turning on and off nodes, and reallocating UM. These two modules are run as events, which are triggered regularly until the simulation finished. This period corresponds to one month in real time. After this period, the *Monitoring* submodule is called to start monitoring the system. This monitoring time corresponds to a time period of two months.

At the end of the simulation, the simulator provides as output the nodes assigned to each user, the QoS of each user, the re-connections of each user (number of times that the information has been replicated), and the average time of MCBR to select a new node to allocate the information. The simulator also allows to take snapshots of different simulations times.

With the objective of applying the MCBR method in Garlanet, we have used three parameters of the system:

- *Node Quality:* It is ranked as the most important parameter. It indicates the node quality, and it is represented as the predicted probability of a node to be connected in a certain period of time. Its value is normalized between 0 and 1, being 1 the highest node quality. To obtain this value, we have generated a prediction model, which is based on the next parameters:
  - *Percentage of connected time:* This parameter indicates the percentage of time the node has been connected (serving) since its first con-

nection. Its value is a normalized percentage between 0 and 1. A node with a high percentage of connected time is desirable, as the stored information is less likely to be reallocated.

– *Number of disconnections:* This parameter complements the previous one and represents the number of times the node has disconnected. Its value is normalized between 0 and 1, taking as upper bound the highest value obtained from a node so far.

Although for the MCBR method the node quality is just a parameter to select the most suitable nodes, with the aim of clarifying how this value is obtained, in next section the highlights of the prediction model are described.

- *Percentage of occupation:* It is ranked as the second most important parameter. This parameter indicates the degree of occupation of a node. It is used to prevent the saturation of nodes. Each node can only host a maximum number of UMs. Its value can be an absolute number between 0 and the maximum number of UMs to host.

- *Download Speed:* It is ranked as the least important parameter. This parameter indicates the download speed of each node. Its value is an absolute number expressed in MB/s.

4. The prediction quality model

This section presents the prediction model used to obtain the quality of a node.

The quality of a node is represented by the likelihood of it keeping connected for a certain period of time. To predict this quality we take into account its behavior during the last month, by obtaining the probability of it being connected for every day of the week. This allows a prediction whether a node will be connected (or not) in the following days.

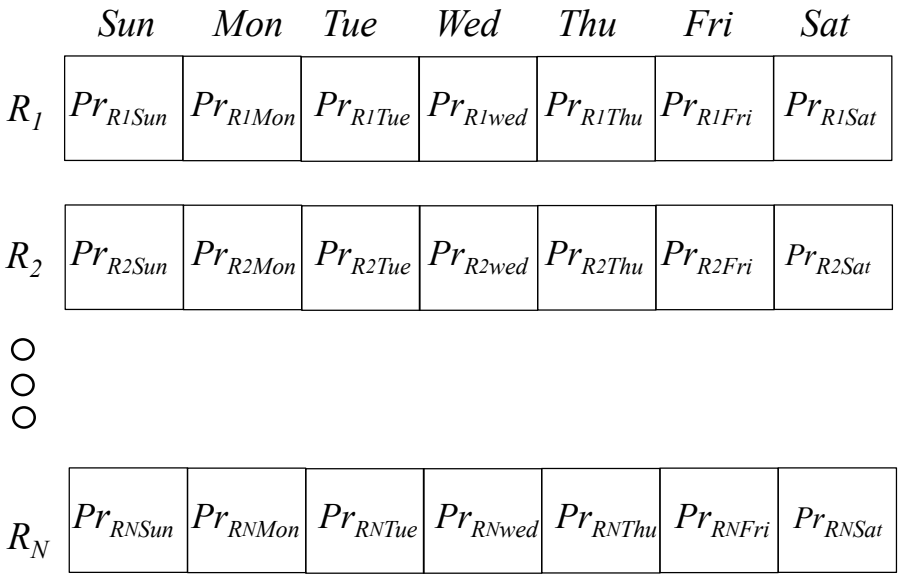


Figure 6: Representation of the data structure for the disconnection probabilities, being n the number of nodes

In order to do so, a data structure is kept for every node which contains the probabilities of disconnection calculated by the prediction model. This structure has the form of an array of seven positions, each of them corresponding to a day of the week. For each day, we store the probability of the pertinent node to disconnect that day, taking into account its disconnection pattern throughout the last four weeks. This data structure is represented in Figure 6, where R_n stands for the node $n \in \{1, 2, ..., N\}$, and $Pr_{R_n,j}$ is the probability of disconnection of the node n the day of the week, $j \in \{1, 2, ..., 7\}$. The intuition behind this procedure represents an effort to learn the habits of the owner of the node, which are relatively consistent from one week to the other.

To record the nodes behavior, that is, the disconnection pattern of each of them over the last four weeks the following procedure is followed.

The data structure of a single node is a four-position matrix, corresponding to each of the four weeks. Each of these indexes contains an array of seven positions, one for each day of the week. This matrix represents the last 28 days

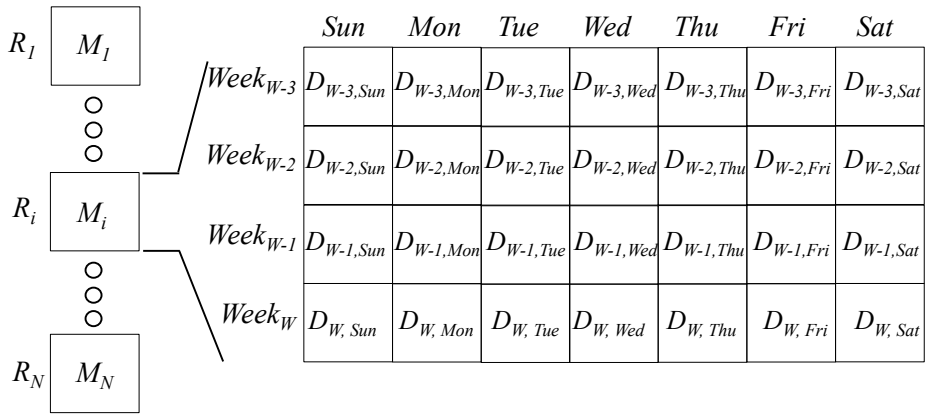


Figure 7: Representation of the data structure used to store the disconnection pattern of the last four weeks of execution for every node

of execution, in which each position will store the number of times the node has moved from a connected to a disconnected state that particular day. In order to update this matrix, every new week an update procedure is performed. This one simply takes all the disconnections from the past seven days of the node, and sums them up for each day of the week in a new seven-position array; then, it removes the oldest recorded week of the node from the matrix, and replaces it with the new one. It then recalculates the probabilities array taking into account all four weeks currently in the matrix. This last step consists, for every day of the week, in getting the average number of disconnections of the past four weeks, and normalizing this value to get a probability of disconnection between 0 and 1. This data structure is represented in Figure 7, where M_i is the four-position matrix corresponding to the node i, W is the number of the last week, and $D_{i,j}$ represents the number of disconnections experienced by the corresponding node of weekday $j \in \{1, 2, ..., 7\}$ from week $i \in \{W-3, W-2, W-1, W\}$. Given this notation, we can define the probability of disconnection of a node n any given weekday j as

$$Pr_{R_n,j} = normalize(\frac{\sum_{w=W-3}^{W} D_{w,j}}{4}) \tag{4}$$

assuming that $D_{w,j}$ is an integer from the corresponding node matrix. The *normalize* function returns a real number between 0 and 1 representing the

Algorithm 1 Disconnection Probability

```
1: procedure discProb(n,x)
2:     nDays ← numbDays(x)                          ▷ Number of days within x
3:     sum ← 0
4:     if x > 0 then
5:         for i ← 1 to i = nDays do
6:             WD_i ← weekDay(today() + i)
7:             sum ← sum + Pr_{R_n,WD_i}
8:     else
9:         for i ← 1 to i = nDays do
10:            WD_i ← weekDay(today() − i)
11:            sum ← sum + Pr_{R_n,WD_i}
12:    return sum/nDays
```

likelihood of disconnection.

Once a new prediction model has been generated, and we have a probability array for every node, we can obtain the probability of a node to be connected in a certain period of time. If we name this requested time t, this is done by calculating two different probabilities and taking the product between them: the disconnection probability of the node for the next t units of time, and the disconnection probability of the node for the past c units of time, where c stands for the time the node has been connected. Using information about the current connection allows for more precise predictions, as the longer a node has been connected, the more likely it is to disconnect in the near future. The obtained result represents a disconnection probability, and therefore has to be subtracted to one in order to get the probability of connection. If we name this probability $Pr_{n,t,c}$, we can formulate it as

$$Pr_{n,t,c} = 1 - (discProb(n,t) * discProb(n,-c)) \tag{5}$$

where $discProb(n,x)$ stands for the function that, given a node n and a lapse of time x, returns the disconnection probability of n in that period. If x is positive, it takes into account, the next x units of time starting with the current day. On the other hand, if x is negative, it considers the previous x units of time. The

Algorithm 2 ILS framework

```
1: procedure ILS
2:     s ← GenerateInitialSolution
3:     bestS ← s
4:     while Stopping condition not met do
5:         s' ← Perturb(s)
6:         s* ← LocalSearch(s')
7:         if ValueObjectiveFuntion(s*) better than ValueObjectiveFunction(bestS) then
8:             bestS ← s*
9:         s ← AcceptanceCriterion(s, s*)
10:    return bestS
```

pseudo-code for this function is shown in Algorithm 1. The function takes the average of their disconnection probabilities stored in the probabilities array for every day of the week found in the specified lapse of time.

5. Metaheuristic algorithm

In order to evaluate the quality of the results provided by the MCBR method, we have developed a metaheuristic algorithm that provides optimal or pseudo-optimal solutions in a reasonably short time. The proposed algorithm is based on the well-known Iterated Local Search (ILS) metaheuristic framework [26]. Algorithm 2 depicts the main components of the ILS framework. First, an initial solution is generated. Then, an iterative process is carried out combining a perturbation stage and a local search stage to improve the initial solution.

The perturbation diversifies the search to be able to escape from local optima. In order to do this, it applies random movements big enough so that the local search cannot undo it in one step. The local search stage aims at searching the best solution inside the neighborhood of the current search space. This procedure consist of exploring the current space performing little changes in the previous solution. Every time a solution which improves the best current solution if found, the best solution is updated. This is done until a predefined stopping condition is met. Then, the best found solution is returned.

Therefore, it is necessary to define each of these stages for the particular problem dealt in this work. The Garlanet scenario and its particularities have been taken into account to propose the algorithm.

For this purpose, first, we depict each node in the network with the attributes explained in section 3: a maximum capacity for hosting UMs, a quality level, and the download speed. All users have a minimum number of replicas of its UMs, and a minimum quality (the sum of the qualities of the nodes assigned to it). Second, we define the goal of the algorithm as the minimization of the total number of replicas of each UM, while ensuring the user restrictions (minimum number of replicas and quality). The reason to consider this objective function is that the lower the number of replicas, the better assignation of nodes have been done, since it involve less movement of data in the system (and therefore better quality of service of the system) while keeping the requirements. Finally, each stage of the ILS framework need to be established for the particular problem: how to generate the initial solution, how to perform the perturbation, and how to perform the local search. Next sections are devoted to explain our proposals in detail.

5.1. Initial solution generation

As mentioned before, the ILS metaheuristic requires an initial solution to start. Therefore, we have defined a method to obtain it. We propose a Multi-Start procedure [27] which executes several instances of the heuristic depicted in Algorithm 3, and chooses the best one as initial solution. This heuristic works as follows. Given the list of nodes decreasingly sorted by their quality, the algorithm assigns UMs to the nodes in the list. Each node is selected randomly by applying BR, using a geometric distribution over the whole list, until the user has reached its minimum demanded quality and the minimum number of required replicas. As stated above, we use a BR selection process to diversify the search space from the beginning. This procedure is applied iteratively for all users. When a node has reached the maximum number of UMs it can hold, it is removed from the sorted list.

Algorithm 3 Heuristic to generate the initial solution

Require:

$nodes \leftarrow Nodes\ list$

$users \leftarrow User\ list$

```
1: procedure ConstructSolution
2:     sortNodes ← nodes.sortDecrByQuality()
3:     users.shuffle()
4:     pointer ← users.first()
5:     while users.notEmpty() do
6:         u ← users.get(pointer)
7:         microservice ← u.getMicroservice()
8:         n ← randomGeom(sortNodes)
9:         assign(n, microservice)
10:        if n.isFull() then
11:            sortNodes.remove(n)
12:        if u.minNodes() ∧ u.minQuality() then
13:            users.remove(u)
14:            pointer ← pointer.next()
```

5.2. Solution perturbation

We propose a perturbation stage where the goal is to change a percentage δ of the total assignations UMs-nodes. Experimentally, after some tests, we have set this parameter to 15%. The algorithm consists of uniformly choosing two random nodes, and trying to move a UM from the first node to the second one. If this is not possible, all potential UMs swaps between both nodes are computed, and a random one is chosen to be performed. This process gives us a modified version of the current solution that we can then be refined to find a local optima.

5.3. Local search

Algorithm 4 describes the implemented local search. Firstly, the users are sorted decreasingly by the total number of replicas of their UMs, and secondly, by their assigned quality. Then, nodes are filtered by all their parameters to

Algorithm 4 LocalSearch

Require:

$nodes \leftarrow Nodes\ list.$

$users \leftarrow Users\ list.$

```
1: procedure LocalSearch
2:     users.sortDecrByMicroservicesAndQuality()
3:     for all param in a Node do                    ▷ Filter Nodes by their parameters
4:         nodes.sortBy(param)
5:         nodes.cutList()
6:     nodes.sortDecrByQuality()
7:     for all u in users do
8:         uNodes ← u.assignedNodes()
9:         uNodes.sortAscendByQuality()
10:        N ← SelectNodesWithHighQualityAvailable(Nodes)
11:        nodesQuality ← 0
12:        IndexToExchange ← 0
13:        for all ni in uNodes do
14:            IndexToExchange ← IndexToExchange + 1
15:            userNode ← ni
16:            nodesQuality ← nodesQuality + userNode.getQuality()
17:            if nodesQuality >= N.getQuality() then
18:                break
19:        N ← assingMicroservice(u.getMicroservice)
20:        uNodes ← ExchangeRespos(N, uNodes[1..IndexToExchange − 1])
```

avoid the worst ones, excluding their quality. Lets k be the number of parameters for every node. Then, we perform k sorting (one for each of the parameters). After this filtering process is done, the nodes are finally sorted by their quality in decreasing order. This leads to a sublist of the best nodes to be used during the assignment.

Afterward, iteratively for each user, we get the first node *(n)* of the list to be exchanged by the largest subset of nodes assigned to the user. If it is not possible to select the first node due to occupancy restrictions, we look for the following node with the highest possible quality. The idea behind it is to

reduce the number of replicas of UMs of each user, while ensuring the minimum demanded quality and nodes.

To do this, we sort the list of nodes already assigned to the user by their quality in an increasing way and iterate over it. During each iteration, we check if we can exchange n by the subset of nodes in the list formed from current iteration node until the one with the lowest quality (first element of the list). When the constraints are violated, node n is exchanged by the subset node list of the previous iteration. Then, the UM of that user is replicated in the node n, the node list and the level of occupancy of the nodes are updated, and finally, the list is sorted for the next user.

An example of this procedure is provided in the following. Consider a $user_i$ which has 4 nodes with the following qualities: 1, 0.8, 0.6, and 0.4, making a total quality of 2.8. On the other hand, we have a node n obtained from the node list with a quality of 1. Supposing that the minimum demanded quality by the users is 2.5, and the minimum number of nodes per user is 3, we can exchange the nodes with qualities 0.6 and 0.4 by n, without violating the mentioned constraints. The algorithm first tries to exchange the node with quality 0.4 by n, which is possible since we would fulfill the constraints of quality and number of nodes. The same holds when it tries to exchange 0.6, 0.4 by n. Then, during the next iteration, it tries to exchange 0.8, 0.6, 0.4 by n, as we we would fulfill the constraints of quality, since the minimum quality allowed is 2.5, we select the sublist of nodes of the previous iteration (0.6, 0.4) to be exchanged by n. Finally, the best solution will be updated if the algorithm value is decreased.

We have set up a maximum time of 150 seconds to find the final solution, since the metaheuristic is able to obtain large improvements at the beginning, but later very small improvements are achieved.

6. Experimental validation

This section is dedicated to assess the performance of MCBR method inside the simulation environment proposed in this paper.

Table 1: Percentages of each kind of nodes in each scenario

	High	**Medium**	**Low**
Scenario 1	5%	20%	75%
Scenario 2	5%	40%	55%
Scenario 3	5%	55%	40%
Scenario 4	10%	45%	45%
Scenario 5	10%	50%	40%
Scenario 6	20%	20%	60%
Scenario 7	20%	40%	40%
Scenario 8	70%	15%	15%

As mentioned, the simulator has been implemented using the programming language Java Standard Edition 7.0. All the computational experiments have been carried out on a workstation with an AMD quad-core processor of 2.3Ghz with 4GB of RAM memory. As operating system we have used CentOS 6.6.

In order to test the behavior of the MCBR method working inside the system, it has been compared with the proposed metaheuristic algorithm. As stated before, this metaheuristic is used to evaluate what would be the near optimal assignment of nodes to users in a idealistic scenario where it can be done from scratch. To allow it, we have taken some snapshots of the simulator when it is working using the MCBR method. With all the information of these snapshots we compare the solution provided by the MCBR method and the solution provided by that metaheuristic in this situation.

Due to the different environments that can appear in these kind of systems, we have simulated different scenarios composed of different percentages of high, medium, and low quality nodes in terms of connection patterns. Table 1 depicts the different tested combinations. We have assumed the most representative scenarios of a real situation. Thus, we have scenarios where *low* and *medium* quality nodes (Scenario 1 to 7) prevail. Moreover, we have consider an "ideal" case (scenario 8), which is more unlikely to appear.

For each scenario the number of nodes has been fixed to 300 and the number of users to 2100. We have considered a realistic scenario where only a 15% of total users of the system contribute by donating free resources. Although other ratios have been tested, e.g., 10% and 5%, the only difference noticed in the results is that the system activity, i.e., connections and re-connections, increases as the ratio decreases. Obviously, if the system is not interesting and the number of users providing resources voluntarily is too low, then the system will fail due to lack of resources. However, this would happen considering any approach.

We have obtained results with 2, 2.5, and 2.75 minimum qualities required for users. We have performed other tests with lower and higher minimum qualities: (a) with a minimum quality lower than 2 users might be at risk of non being able to access the service; (b) a minimum quality higher than 2.75 may produce the situation where there are not enough available nodes to be assigned in order to guarantee that minimum quality.

With the described scenarios and parameters, we have considered different performance indicators for the algorithms. The choice of these indicators has been made considering the possible failures or weaknesses in the system.

One of the most important features regarding the algorithm to use in the system is the computational time it requires. Since the system works in real time, the sooner it provides the output, the better it works. While the algorithm is running, system changes occur (e.g. new connections and disconnections) and the algorithm result may not be appropriate. Figure 8 shows a comparison between the proposed algorithm working inside the simulator (MCBR) and the proposed metaheuristic (MET) for each different minimum quality tested (MQ). The box-plots depict the distribution of the computational times obtained for the different considered scenarios. As can be checked, the times required by the metaheuristic are too large (around 150 seconds) to be considered in a real time system. On the contrary, the proposed MCBR method only needs an average of 2 seconds to provide a result, which seems more appropriate for the system.

Regarding the quality offered to each user of Garlanet once the minimum

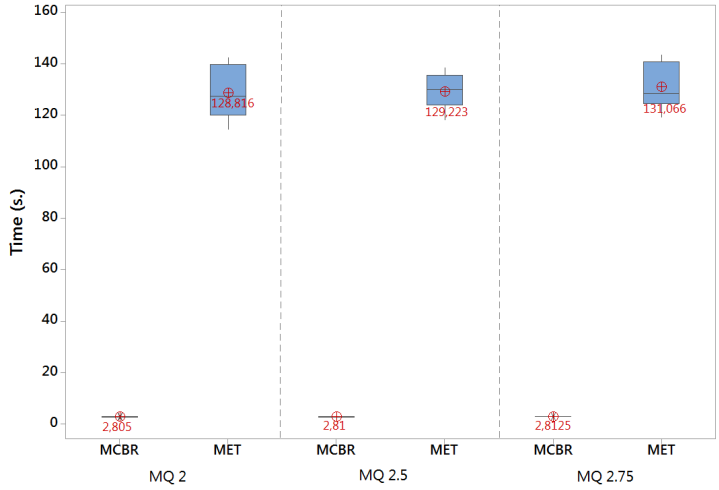


Figure 8: Computational times

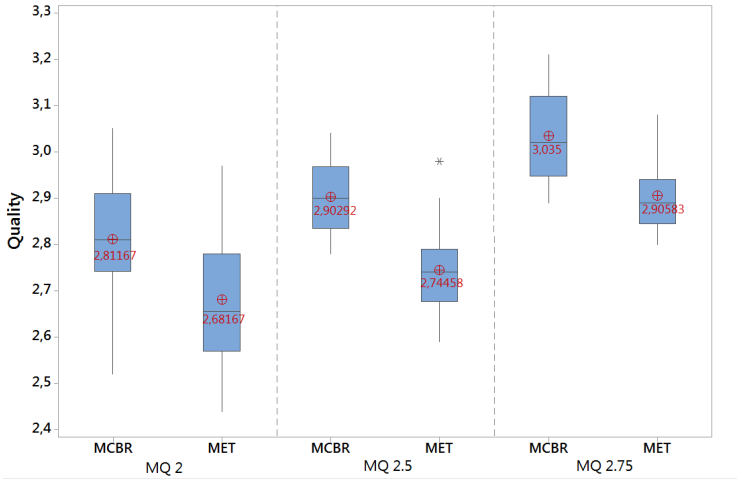


Figure 9: Quality assigned to each node in the network

quality is fixed (i.e., 2, 2.5, 2.75), we have noticed that the metaheuristic is able to better fit it, since it takes into account the whole system at once and can manage qualities better. The MCBR method tends to provide more quality than the minimum required because it has to work with the remaining free nodes, as highlighted in Figure 9.

Due to the way the MCBR method works, when a node disconnects the

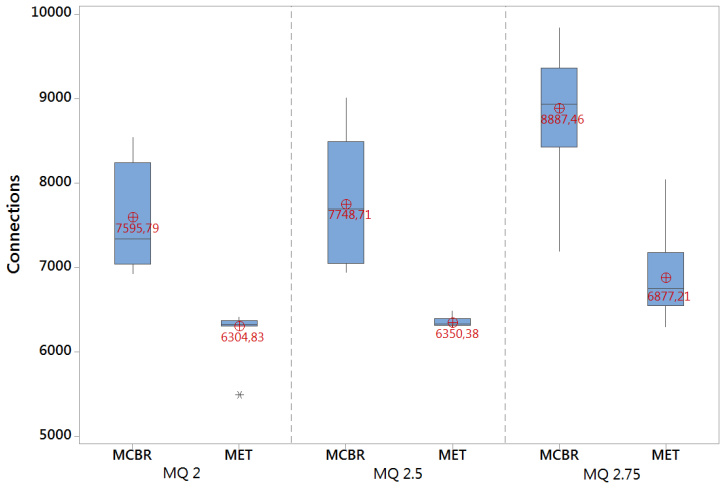


Figure 10: Number of connections between nodes and nodes

CSS needs to find new available nodes that, together with the ones assigned to each user, allows to reach the minimum quality for each of them. Thus, the connections between the users and their nodes are kept and new ones are added to reach the minimum quality, while the metaheuristic tries to find the best group of nodes to be assigned to each user without taking into account the currently assigned ones. For this reason, in Figure 10 we can see that the metaheuristic is able to obtain a best combination of connections between users and nodes to reach the minimum quality, while the MCBR method needs more connections (with an increase of 30%).

However, the MCBR method is able to avoid the movement of a large quantity of information to be copied between nodes within the network, so that time and network overload are saved. This kind of network is devoted to serve the users of the application, and information flows as copies could slow down and reduce the quality of service provided to users. In this regard, Figure 11 shows the number of copies or re-connections needed if a node disconnects for the MCBR method and the metaheuristic algorithm. As can be checked, the number of copies when applying the metaheuristic is so high (more than 6 times the number of copies with the MCBR method) that it is impossible to think

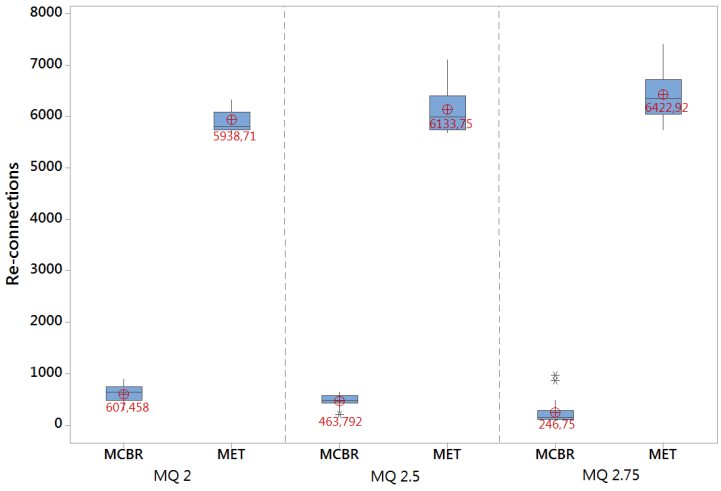


Figure 11: Number of re-connections needed when a node disconnect

about using it inside a real system.

Therefore, we have verified that the MCBR method is a fast method and is working properly, providing the quality required to each user of the application through a set of assignments of users to nodes that guarantees a correct functionality of the system, i.e., avoiding excess of data movement around the network and servicing all users.

Regarding the minimum quality to offer, it seems that a quality of 2.5 provides results with a trade-off among the different dimensions considered: average time, average connections, average re-connections, and average final quality. Figure 12 is a visual representation of the solutions obtained when using the different minimum qualities (MQ). For each dimension, it shows averages. On the one hand, when the minimum quality is 2, we can guarantee the smallest number of connections and short computational times. However, this involves the highest number of re-connection and, therefore, movement and copy of data. On the other hand, when the minimum quality is 2.75 the situation is the opposite, i.e., the algorithm needs more computational time and, although we can guarantee a low level of re-connections, the number of nodes servicing each user is high. In this case, the system needs to keep many nodes updated and leads to

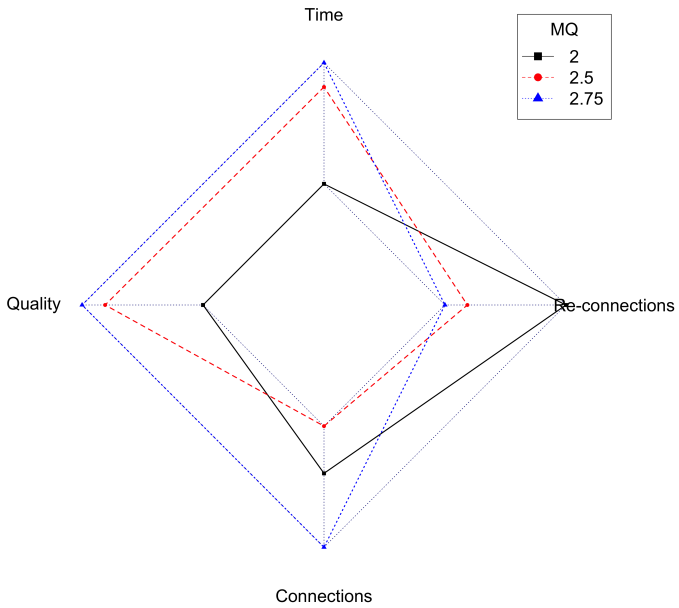


Figure 12: Comparative of the different dimensions for solutions obtained using different minimum qualities

a high flow of data through the network. For these reasons, a minimum quality 2.5 is chosen, which presents a balance among the different dimensions.

7. CONCLUSIONS

In this work we have presented and validated the MCBR method, which is a generic selection method based on a multi criteria optimization strategy to select the most suitable nodes in VC distributed systems. MCBR strives to provide high quality solutions in a fast way, using very low computing times. Due to its flexibility, the MCBR can be applied in a wide range of distributed systems.

As we have shown in the experimental validation section, MCBR provides good quality solutions in a fast way, under the consideration of different user constraints, such as the minimum number of nodes and QoS. Moreover, MCBR avoids the excessive data movement within the network. A metaheuristic has

been used to check the quality of our results. Although this metaheuristic is able to improve the results obtained with our MCBR method, due to the inherit time constraints of a real-time system and the volume of data that this metaheuristic involves, it is not possible to apply it inside the system.

As future work, we plan to extend the current method taking into account the data location to select the most suitable nodes. Data location has a significant impact on several network performance criteria. For example, placing data near users may reduce the network congestion and improve the load balancing. Therefore, through this extension, we planing to consider a trade-off between the node quality and its location within the network.

Acknowledgements

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness (TRA2013-48180-C3-P, TRA2015-71883-REDT), FEDER. and the Erasmus+ programme (20161ES01KA108023465).

References

[1] M. N. Durrani, J. A. Shamsi, Volunteer computing: requirements, challenges, and solutions, Journal of Network and Computer Applications 39 (2014) 369 – 380.

[2] R. Marler, J. Arora, Survey of multi-objective optimization methods for engineering, Structural and Multidisciplinary Optimization 26 (6) (2004) 369–395.

[3] A. A. Juan, J. Faulin, J. Jorba, D. Riera, D. Masip, B. Barrios, On the use of Monte Carlo simulation, cache and splitting techniques to improve the Clarke and Wright savings heuristics, Journal of the Operational Research Society 62 (2010) 1085–1097.

[4] Garlanet, available at `http://http://dpcs.uoc.edu/projects/garlanet`.

[5] I. Boussad, J. Lepagnot, P. Siarry, A survey on optimization metaheuristics, Information Sciences 237 (2013) 82 – 117.

[6] T. Estrada, O. Fuentes, M. Taufer, A distributed evolutionary method to design scheduling policies for volunteer computing, in: Proceedings of the 5th Conference on Computing Frontiers, CF '08, 2008, pp. 313–322.

[7] T. Ghafarian, H. Deldari, B. Javadi, M. H. Yaghmaee, R. Buyya, Cycloidgrid: A proximity-aware p2p-based resource discovery architecture in volunteer computing systems, Future Generation Computer Systems 29 (6) (2013) 1583 – 1595.

[8] T. Ghafarian, B. Javadi, Cloud-aware data intensive workflow scheduling on volunteer computing systems, Future Generation Computer Systems 51 (2015) 87 – 97.

[9] S. Sebastio, G. Gnecco, A. Bemporad, Optimal distributed task scheduling in volunteer clouds, Computers & Operations Research 81 (Supplement C) (2017) 231 – 246.

[10] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, Distributed optimization and statistical learning via the alternating direction method of multipliers, Found. Trends Mach. Learn. 3 (1) (2011) 1–122.

[11] H. Guler, B. B. Cambazoglu, O. Ozkasap, Task allocation in volunteer computing networks under monetary budget constraints, Peer-to-Peer Networking and Applications 8 (6) (2015) 938–951.

[12] T. Duong-Ba, T. Nguyen, B. Bose, D. A. Tran, Distributed client-server assignment for online social network applications, IEEE Transactions on Emerging Topics in Computing 2 (4) (2014) 422–435.

[13] L. Zhang, X. Tang, The client assignment problem for continuous distributed interactive applications: Analysis, algorithms, and evaluation, IEEE Trans. Parallel Distrib. Syst. 25 (3) (2014) 785–795.

[14] C. Diot, L. Gautier, A distributed architecture for multiplayer interactive applications on the internet, IEEE Network 13 (4) (1999) 6–15.

[15] H. Zheng, X. Tang, The server provisioning problem for continuous distributed interactive applications, IEEE Transactions on Parallel and Distributed Systems 27 (1) (2016) 271–285.

[16] H. Nishida, T. Nguyen, Optimal client-server assignment for internet distributed systems, in: 2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN), 2011, pp. 1–6.

[17] B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman, E. Levy, A. Maraschini, P. Massonet, H. Munoz, G. Tofetti, Reservoir - when one cloud is not enough, Computer 44 (3) (2011) 44–51.

[18] R. de C. Coutinho, L. M. A. Drummond, Y. Frota, Optimization of a Cloud Resource Management Problem from a Consumer Perspective, 2014, pp. 218–227.

[19] T. A. Feo, M. G. C. Resende, Greedy randomized adaptive search procedures, Journal of Global Optimization 6 (2) (1995) 109–133.

[20] R. d. C. Coutinho, L. M. Drummond, Y. Frota, D. de Oliveira, Optimizing virtual machine allocation for parallel scientific workflows in federated clouds, Future Gener. Comput. Syst. 46 (2015) 51–68.

[21] L. Heilig, E. Lalla-Ruiz, S. Vo, A cloud brokerage approach for solving the resource management problem in multi-cloud environments, Computers & Industrial Engineering 95 (2016) 16 – 26.

[22] A. D. Gonçalves, L. M. de A. Drummond, A. A. Pessoa, P. M. Hahn, Improving lower bounds for the quadratic assignment problem by applying a distributed dual ascent algorithm, CoRR abs/1304.0267.
URL `Availablefrom:http://arxiv.org/abs/1304.0267`

[23] X. Serra, J. de Armas, J. M. Marqus, Simulating and optimizing resource allocation in a micro-blogging applicacion, in: 2016 Winter Simulation Conference (WSC), 2016, pp. 3167–3176.

[24] A. Grasas, A. A. Juan, J. Faulin, J. de Armas, H. Ramalhinho, Biased randomization of heuristics using skewed probability distributions: A survey and some applications, Computers & Industrial Engineering 110 (Supplement C) (2017) 216 – 228.

[25] C. A. R. Hoare, Algorithm 64: Quicksort, Commun. ACM 4 (7) (1961) 321–331.

[26] H. R. Loureno, O. C. Martin, T. Sttzle, Iterated local search, in: Handbook of Metaheuristics, volume 57 of International Series in Operations Research and Management Science, 2002, pp. 321–353.

[27] L. Kocsis, A. György, Efficient multi-start strategies for local search algorithms, in: Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2009, Bled, Slovenia, September 7-11, 2009, Proceedings, Part I, 2009, pp. 705–720.