Heuristic Performance Evaluation for Load Balancing in Cloud

Bruno G. Batista,
Natan B. Morais,
Bruno T. Kuehne,
Rafael M. D. Frinhani
Federal University of Itajubá (UNIFEI)
Itajubá, MG - Brazil
Email: {brunoguazzelli, morais_natan,
brunokuehne, frinhani}@unifei.edu.br

Dionisio M. L. Filho
Federal University of
Mato Grosso do Sul (UFMS)
Ponta Porã-MS, Brazil
Email: dionisio.leite@ufms.br

Maycon L. M. Peixoto
Federal University of Bahia (UFBA)
Salvador-BA, Brazil
Email: mayconleone@dcc.ufba.br

***Abstract*—Cloud computing introduces a new level of flexibility and scalability for providers and clients, because it addresses challenges such as rapid change in Information Technology (IT) scenarios and the need to reduce costs and time in infrastructure management. However, to be able to offer quality of service (QoS) guarantees without limiting the number of requests accepted, providers must be able to dynamically and efficiently scale service requests to run on the computational resources available in the data centers. Load balancing is not a trivial task, involving challenges related to service demand, which can change instantly, performance modeling, deployment and monitoring of applications in virtualized IT resources. In this way, the aim of this paper is to develop and evaluate the performance of different load balancing heuristics for a cloud environment in order to establish a more efficient mapping between the service requests and the virtual machines that will execute them, and to ensure the quality of service as defined in the service level agreement. By means of experiments, it was verified that the proposed algorithms presented better results when compared with traditional and artificial intelligence heuristics.**

Index Terms*—Cloud Computing; Load Balancing; Performance Evaluation.

I. Introduction

In recent years, cloud computing has been one of the most widely discussed topics in IT (Information Technology). According to NIST (National Institute of Standards and Technology), "Cloud computing is a model that allows ubiquity, convenience and on-demand access to a shared pool of configurable resources and can be quickly delivered with minimum management effort on the part of the users" [9].

However, as the cloud is a distributed system that provides services on-demand and in a transparent way, the computational system present in this environment is expected to operate appropriately. To achieve this, the system must provide a suitable performance both in terms of response time and availability (to minimize the risk of disrupting the service being offered), and security (to avoid loss of data or messages) so that it can attract the confidence of its clients and give them the satisfaction they expect. Hence, service providers must ensure that the different attributes of Quality of Service (QoS) are satisfied.

Ensuring QoS in a cloud environment is not a trivial task, since there are different types of users and devices with various service requirements [10]. Furthermore, different providers may offer the same service by deploying different technologies. The QoS settings begin by establishing the parameters required by the users. These parameters are mapped and negotiated between the components of the system to ensure that everyone is able to achieve an acceptable level of QoS, and thus define a Service Level Agreement (SLA). After the terms of the SLA have been defined, the resources are allocated and monitored, with the possibility of a renegotiation if the conditions of the system change [1].

In this way, the QoS and compliance with SLAs have been topics of great interest in recent years in both academia and industry. These topics involve several challenges, such as users and service characterization, admission control, prediction, analysis and load balancing, monitoring, resources provisioning, systems optimization, among others. Considering these challenges, the focus of this study involves cloud load balancing.

The demand for services can vary unpredictably, increasing or reducing the number of service requests. For the user, all the complexity behind a cloud environment is transparent, since he is concerned only with the availability, access, performance, and cost of a service. The provider, in turn, must ensure that this demand is met and allocated for execution in the virtualized computing resources in order to comply with the SLA, as well as the efficient use of resources, at a fair price. For this, efficient load balancing algorithms must be considered.

Therefore, the study carried out in this paper aims to develop and evaluate load balancing heuristics for a cloud environment, in order to guarantee compliance with service level agreements, as well as efficient use of computational resources. These algorithms must consider basic premises present in traditional algorithms, as well as techniques of optimization and artificial intelligence.

All the material discussed in this paper is structured in the following sections: the state-of-the-art is sets out in Section

978-1-5386-7879-4/18/$31.00 ©2018 IEEE
DOI 10.1109/HPCS.2018.00099

II; Section III describes the developed algorithms; Section IV presents the design of the experiments; the experimental results are analyzed in Section V; finally, Section VI presents the conclusions as well as the premises for future work.

II. State-of-the-art

The unpredictability and constant behavior change in a cloud computing environment coupled with the ability to offer to users the highest possible satisfaction in providing a service demonstrates the importance of load balancing [7]. The load balancing can be described as responsible for distributing service requests between physical and virtual resources in order to avoid under- or over-utilization of available resources, as well as guaranteeing the user the execution of the SLA [1].

However, according to [11], the application of load balancing mechanisms involves certain challenges, of which the following stand out:

- **Reliability:** the chosen technique must be reliable, so that there are no errors and modifications in the user data, since this data will be transferred from one location to another during the execution of the balancing.
- **Adaptability:** load balancing must adapt to the state of the environment, changing the way of operating when encountering adverse situations and assigning the loads to the computational resources in the shortest time possible, always respecting the SLA.
- **Fault Tolerance:** the algorithm must handle exceptions. In this way, when encountering an atypical situation and/or errors during execution, it must continue operating.
- **Throughput:** the technique must ensure that there is the highest throughput possible at the lowest expense. If an algorithm does not increase system throughput, it does not fulfill its purpose.
- **Waiting Time:** the waiting time of a request to be allocated to a resource should be minimized whenever possible.

There are several studies in the literature that deal with load balancing. In [7], the authors discuss the importance of load balancing in a cloud environment. They present the Min-Min and Max-Min techniques, exploring and comparing how these traditional approaches act. In the performed experiments, it was verified that the processing time is smaller using the Max-Min technique.

Tyagi and Kumar [12] discuss about three different methods of load balancing: Round Robin, method based on time division in quantum; the Equally Spread Current Execution Location, which provides resources randomly for the requests processing taking into account the idleness of the resource; and Throttled Load Balancing Policy, which defines the order of resource adequacy for each type of request, in which if all are busy, the request is inserted in a queue and waits to be executed. An analysis of these methods was performed considering the response mean time as the response variable. According to the results, the method Throttled proved to be more efficient in relation to the others.

Singh et al. [11] cite possible forms of load balancing: the Static, which is responsible for a small cloud environment, with fast Internet and no focus on communication delay; the Dynamic, focused on reducing the communication delay and execution time, as well as being appropriate for larger environments; and the Hybrid, which uses a combination of both previous concepts. In this way, the analyzed environment was based on a dynamic balancing, which was analyzed by the authors using the method of autonomic agents. Of the three agents present, two of them used natural computing, where the ant colony heuristic was adopted. Analyzes verified that this mechanism proved to be an efficient form of balancing, both in times of under- and over-utilization of computational resources.

In the study presented in [5], the authors discuss a totally different way of dealing with the load balancing problem. They proposed a method that uses a genetic algorithm, which follows the basic principles of initial population, cross-over between genes and mutation. In the experiments, the authors used a simulated environment with different numbers of Data Centers available to verify and compare the methods of Round Robin, First Come First Serve, Stochastic Hill Climbing and the proposed Genetic Algorithm. In this analysis, it was possible to verify that the greater the availability of virtual machines (VMs) in a Data Center, the greater the effectiveness of the Genetic Algorithm in comparison to the others. It is worth mentioning that Stochastic Hill Climbing, another artificial intelligence technique, obtained better results than the other algorithms in practically all the experiments, behind only the Genetic Algorithm.

In [13], the authors did not propose any new method, but presented an extension to the scheduling algorithm called HySARC^2. This algorithm is composed of three parts: i) analysis of available resources and clustering; ii) provisioning of different groups of tasks for different clusters; and iii) scheduling of requests to previously defined clusters. The authors proposed a change based on a hybrid performance, with two different algorithms for scheduling. Thus, HySARC^2 can work with different algorithms, at the same time, according to the need of each group of requests and resources.

Based on the analyzed papers, it was possible to analyze the characteristics of each heuristic, as well as the shortcomings of each approach, aiming the development of new algorithms capable of dealing with different service demands. These algorithms are described in the next section.

III. Algorithms

The purpose of this study was to implement and analyze the performance of distinct load balancing algorithms in a cloud environment. For this, it was necessary to use a cloud environment for the proposed implementations, analyzes and validations. Thus, the study developed in [2] was used as a basis, in which the authors proposed the ReMM (Resource Management Module).

ReMM is a cloud resource management module, initially implemented in the CloudSim simulator [4], which consid-

ers the manipulation of computational resources on-the-fly, respecting the quality of service metrics defined in the service level agreement. It aims to meet any request demand, applying both horizontal and vertical scalability in order to dynamically change the amount of computational resources, impacting the price. Considering the paper limitation, more information about ReMM can be find in [2] and [3].

In the current version, only the First Come First Served (FCFS) algorithm is available in ReMM to perform the load balancing process. Thus, considering the great dynamicity of service demand, new algorithms were developed and inserted in ReMM in the study carried out in this paper. The following sections describe these algorithms.

A. First Come First Served - FCFS

FCFS works with a list of requests. The requests arrive and are answered in a sequential way by the resources available in the system, that is, the first incoming request will be assigned to the first resource, the second request to the second resource and so on until the end of the executions. Upon arriving at the last resource, when a new request arrives, it will be assigned to the first resource used, resetting the resource list. Thus, the sequential assignment of the requests to the virtual machines, which are organized in a circular list, is applied [5].

B. Min-Min

The Min-Min algorithm is a balancing strategy that aims to execute as many requests as possible, that is, to maximize the throughput. To do this, it sends all small requests to the most powerful resources and large requests for resources with the lowest computational power. In this way, the requests will be executed faster, allowing other smaller requests to be sent to these resources. With this strategy the most powerful resources are not at any time occupied with larger and more expensive requests and always manage to maintain the fluidity of the system, quickly performing all the tasks assigned to them [7].

On the other hand, larger and more expensive requests are delivered to the less powerful resources of the environment, leading in a longer time for them to be finalized. This also makes the possibility of starvation greater, as it may occur when all resources with lower computational power are busy with large requests, and new requests of this size arrive in the system requiring to be executed. In this way, they must wait for some of these resources to be idle, which can take a long time.

C. Ants Colony

The Ant Colony heuristic is an optimization algorithm generally used when dealing with graph theory and resolutions of complex and known problems, as the shortest path problem. However, it can also be used for problems such as load balancing, as shown in [14].

The ant colony, as its name says, is a technique based on the ants behavior. Its heuristic is based on probability, and because it is an intelligent algorithm, it needs to improve over time, learning new things, misunderstanding and accepting mutations, in order to find the best overall solution to the problem.

The algorithm execution begins when the ant leaves its colony in search of food, making its journey almost randomly, since it does not know where the food is. If the food is found, the ant returns doing the same route of the way releasing pheromone during its journey. In this way, when other ants go in search of food, the search will no longer be as aimless, since they already have pheromones to follow [6].

This search occurs through the pheromones of other ants, but it is not the only path that must be followed. The ant is likely to choose another path or to miss the path the other ant made. If the path is worse than the previous one, over time the amount of pheromone will decrease (rate of evaporation). However, by finding better and more efficient paths, more ants will use them, making the pheromone increase, which results in a greater probability of using this path.

D. Dynamic Min-Min

Dynamic Min-Min is an algorithm based on Min-Min and proposed in this study. This new approach follows the main precept of the original algorithm, which is the execution of small requests in more powerful resources and larger requests in less powerful resources. However, this algorithm adopts a dynamic approach. Its operation consists in the organization of the computational resources of the environment and the requests in different lists, according to the computational power and the size of the request, respectively.

Differently from the original Min-Min algorithm, Dynamic Min-Min does not split the resources into just two equal-sized groups. It analyzes the environment and separates available resources into groups that contain only resources with the same computational power. For example, if an environment contains 20 weak VMs, 40 medium VMs, and 30 strong VMs, then three clusters with 20, 40 and 30 VMs, respectively, will be created, each one containing only the resources of the same group.

Then, Dynamic Min-Min analyzes and classifies a request according to its size. In the original algorithm, the request is analyzed and classified as a small or large request in a static way, i.e., before the environment execution. In the new algorithm, this analysis and classification occurs dynamically, on-the-fly, allowing the definition of different lists with distinct groups of requests.

In this way, a more assertive distribution is applied, since a more intrusive and dynamic analysis is applied in the computational resources and in the requests, allowing the more efficient grouping and mapping between requests and resources.

E. Dynamic Distribution for Efficient Use of resources - DDEU

DDEU is another algorithm proposed in this paper. It applies a strategy of dynamic allocation of requests based on the idleness of resources of a particular computational power

group, in order to meet the deadline defined in the SLA and the efficient use of resources.

Initially, there is the warm up phase, in which each available computational resource receives a request of any size to execute, following the methodology applied in FCFS. Thus, at this stage the characteristics of requests and computational resources are not considered, allowing any request to be sent for execution in any resource.

After the warm up, the computational resources are divided into groups according to computational power. Within each group, resources are organized into a list according to the utilization rate of each one. As soon as a new request arrives, it is sent to the group that has the highest probability to execute it within the deadline defined in the SLA. Within the group, it is analyzed whether the first computational resource of the list is idle. If it is, it receives the request for execution. Otherwise, the next resource in the list is parsed.

All heuristics described in this section were inserted and evaluated using ReMM. The next section presents the design of the experiments.

IV. Design of the Experiments

For the execution of the experiments, a provider composed by data centers was configured, which hosted the virtual machines that executed the service requests. The physical resources were considered unlimited, and thus, all requests were accepted and executed by the VMs. In addition, there was no differentiation of users through priorities.

The virtual machines were modeled based on the M3 instances types of Amazon[1]. Table I shows the settings for each VM.

TABLE I
Specification of instances.

Instances	Virtual core	Main memory (GB)	Disk SSD
m3.medium	1	3.75	1 x 4
m3.large	2	7.5	1 x 32
m3.xlarge	4	15	2 x 40
m3.2xlarge	8	30	2 x 80

Real-time, lightweight and heavyweight requests were dynamically created and sent during simulation observation time. This type of submission made it possible to create more realistic simulations because it made the user behavior closer to what is expected in a real environment[2]. A SLA was defined, which stipulated a deadline of 1000 milliseconds to execute a request, with a deviation of 15%.

The experiments presented in the next section were conducted with the purpose of evaluating metrics related to service performance using different load balancing algorithms in the provider. Thus, the following response variables were considered:

- **Execution Mean Time (EMT):** the average time spent, in milliseconds, in the execution of service requests during the observation time.
- **Number of Answered Requests (AR):** average rate of requests met by a cloud environment during the observation time.

A full factorial design was used following the methodology presented by [8], in which the factors correspond to the characteristics of the analyzed environment and the levels are the possible variations that the environment can present. Thus, 2 factors and their respective levels were considered and presented in Table II.

TABLE II
Factors and levels.

Factors	Levels
Algorithm	FCFS, Min-Min, Dynamic Min-Min, Ant Colony or DDEU
Instance	*m3.medium*, *m3.large*, *m3.xlarge* or *m3.2xlarge*

In Table II, the Algorithm factor defines whether requests, which may be light or heavy, will be sent to execute in available instances (*m3.medium*, *m3.large*, *m3.xlarge* or *m3.2xlarge*) by means of the FCFS, Min-Min, Dynamic Min-Min, Ant Colony or DDEU balancing algorithms.

Another important factor is the number of VMs available to meet the services demand. For this factor, a fixed value of 500 VMs[3] was defined, which were grouped according to the cases defined (Cases 1, 2, 3 and 4):

- **Case 1 - Equal Division:** of the 500 VMs available in the environment, 125 belong to each of the four types presented in Table I. Therefore, a cloud environment with the same amount of VMs for all types is analyzed in this case.
- **Case 2 - Predominance of VMs with Low Processing Power:** in this case most of the environment concentrates VMs with low processing power, i.e., 200 VMs are *m3.medium*, 200 are *m3.large*, 50 are *m3.xlarge* and 50 are *m3.2xlarge*. Thus, 80% of the VMs available in the environment are of the two types with lower computational power.
- **Case 3 - Predominance of VMs with Intermediate Processing Power:** for this case there is the predominance of VMs with intermediate processing power, using more instances of *m3.large* and *m3.xlarge* types, with 200 VMs each one. For both *m3.medium* and *m3.2xlarge* types, there is a total of 100 VMs available, 50 VMs for each type.
- **Case 4 - Predominance of VMs with High Processing Power:** finally, there is the case where 200 VMs are *m3.xlarge*, 200 VMs are *m3.2xlarge* and only 50 VMs are allocated for each of the two instances types with lower processing value (*m3.medium* and *m3.large*). In this way, an environment with greater accumulated processing power is modeled.

Therefore, a cloud system was simulated with 500 VMs, which were organized in different quantities according to the

[1]https://aws.amazon.com/en/ec2/instance-types/

[2]Other types of loads can be exploited in future work.

[3]Other configurations and quantities of VMs can be explored in future works.

case, allowing evaluations of the balancing algorithms in the predominance of different types of virtual machines.

In addition, two experiment scenarios were defined. For the first scenario the analyzed algorithms were FCFS, Min-Min and Dynamic Min-Min (proposed in this paper). These algorithms were chosen so that the behavior of Dynamic Min-Min could be analyzed in relation to traditional algorithms, which are very well documented in the literature.

In the second scenario, the analyzed algorithms were Dynamic Min-Min, Ant Colony and DDEU (also proposed in this paper). Dynamic Min-Min was again chosen so that its performance is now not compared to traditional algorithms, but rather to more complex and intelligent algorithms.

In both scenarios, each experiment was run 10 times. Through 10 replications it was noted that the results for each response variable did not show significant variations. Thus, the average, standard deviation and the interval confidence of 95% were calculated for each setting of the experiments. Finally, for the first scenario, an observation time of approximately 86,400 seconds was defined, corresponding to 1 day of uninterrupted execution, while for the second scenario the observation time was 172,800 seconds, i.e., 2 uninterrupted days of execution. The need for an even longer observation time is due to the fact that Ant Colony and DDEU are intelligent algorithms that need a longer time to reach convergence.

V. Analysis of the Results

From the planning specified in the previous section, it was possible to model an experimentation environment with different VMs clusterings, allowing to compare the balancing algorithms through the response variables. Thus, two scenarios of experimentation were defined and analyzed.

A. Scenario 1

Figure 1 and Table III present the experiments results considering the requests execution mean times (EMTs), in milliseconds, during the simulation observation time. It was verified that the Dynamic Min-Min algorithm obtained the shortest times for all the analyzed cases, except for the fourth case, since the algorithm tried to maintain the requests execution times as close as possible to the defined deadline.

For Case 1, in which the number of virtual machines *m3.medium*, *m3.large*, *m3.xlarge* and *m3.2xlarge* was the same, the Dynamic Min-Min heuristic reduced the requests EMTs by approximately 15% when compared to the FCFS algorithm and 51% over Min-Min. This reduction was approximately 56% and 88%, respectively, for the experiments with the premises defined in Case 2, in which there was a predominance of VMs with lower computational power (*m3.medium* and *m3.large*). For Case 3, in which there was predominance of VMs with intermediate computational power(*m3.large* and *m3.xlarge*), there was a reduction in the EMT of approximately 2% with the Dynamic Min-Min algorithm in relation to FCFS and of 28% with the Dynamic Min-Min in relation to Min-Min. Finally, in Case 4, in which the number of VMs *m3.xlarge* and *m3.2xlarge* was predominant (80% of the total), the Dynamic Min-Min algorithm presented requests execution mean times larger than those obtained in the FCFS and Min-Min algorithms, approximately 25% and 7%, respectively. This occurred because Dynamic Min-Min aims to maximize system throughput by assigning the highest requests for the least powerful resources and the smallest requests for the most powerful resources dynamically. However, with the predominance of more powerful instances in Case 4, heavy requests overloaded less powerful machines (accounting for 20% of total VMs), generating queues of large requests. Consequently, the execution time of these requisitions has been impaired. In addition, the Dynamic Min-Min algorithm takes into account the deadline defined in the SLA to adjust itself and try to ensure the required QoS, as opposed to FCFS and Min-Min algorithms.

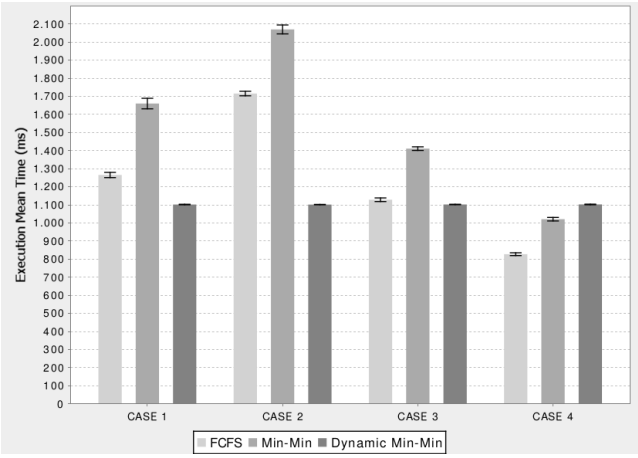

Fig. 1. Response mean times in Scenario 1.

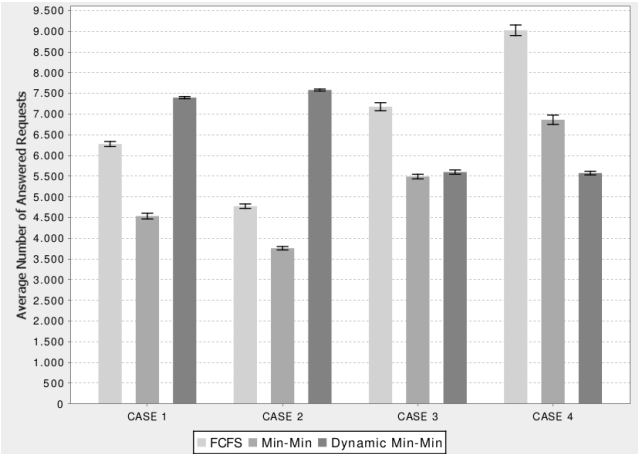

Fig. 2. Average number of answered requests in Scenario 1.

Considering the average number of answered requests, the lower the time spent executing a request, the greater the throughput of the environment. In Figure 2 and Table IV it is possible to note that, under conditions of equality between

TABLE III
COMPARISON OF THE EXECUTION MEAN TIMES.

Case	Algorithm	EMT (ms)	Standard Deviation	Interval Confidence	Lower Limit	Upper Limit
Case 1	FCFS	1265.24	23.27	14.42	1250.81	1279.66
Case 1	Min-Min	1660.14	47.05	29.16	1630.98	1689.31
Case 1	Dynamic Min-Min	1102.07	2.37	1.47	1100.60	1103.54
Case 2	FCFS	1715.50	20.26	12.55	1702.95	1728.06
Case 2	Min-Min	2069.97	39.83	24.69	2045.28	2094.65
Case 2	Dynamic Min-Min	1101.76	1.46	0.91	1100.85	1102.67
Case 3	FCFS	1128.52	16.05	9.95	1118.57	1138.47
Case 3	Min-Min	1411.00	15.96	9.89	1401.11	1420.89
Case 3	Dynamic Min-Min	1102.31	2.03	1.26	1101.06	1103.57
Case 4	FCFS	827.47	12.43	7.71	819.76	835.17
Case 4	Min-Min	1020.81	15.76	9.77	1011.04	1030.58
Case 4	Dynamic Min-Min	1102.46	2.75	1.70	1100.76	1104.16

TABLE IV
COMPARISON OF THE AVERAGE NUMBER OF ANSWERED REQUESTS.

Case	Algorithm	AR	Standard Deviation	Interval Confidence	Lower Limit	Upper Limit
Case 1	FCFS	6277.60	97.99	60.73	6216.87	6338.33
Case 1	Min-Min	4535.80	111.73	69.25	4466.55	4605.05
Case 1	Dynamic Min-Min	7398.70	43.07	26.69	7372.01	7425.39
Case 2	FCFS	4775.60	85.86	53.22	4722.38	4828.82
Case 2	Min-Min	3759.60	65.99	40.90	3718.70	3800.50
Case 2	Dynamic Min-Min	7582.00	38.29	23.73	7558.27	7605.73
Case 3	FCFS	7179.10	155.59	96.44	7082.66	7275.54
Case 3	Min-Min	5491.20	90.72	56.23	5434.97	5547.43
Case 3	Dynamic Min-Min	5599.00	83.08	51.49	5547.51	5650.49
Case 4	FCFS	9026.10	205.41	127.31	8898.79	9153.44
Case 4	Min-Min	6864.10	181.30	112.37	6751.73	6976.47
Case 4	Dynamic Min-Min	5574.90	70.80	43.88	5531.02	5618.78

the number of VMs of different types (Case 1), the Dynamic Min-Min algorithm answered more requests when compared to FCFS and Min-Min heuristics, approximately 15% and 39%, respectively. In Case 2, with predominance of less powerful instances, the Dynamic Min-Min algorithm also proved to be more efficient, increasing the average number of answered requests in 37% when compared to FCFS and 50% to Min-Min. However, with the computational power increase in Cases 3 and 4, the FCFS algorithm presented better results considering the average number of answered requests. For Case 3, Dynamic Min-Min answered approximately 28% less requests than FCFS and only 2% more requests than Min-Min. Finally, in Case 4 (predominance of more powerful instances), Dynamic Min-Min obtained the worst result in relation to all analyzed cases, a reduction of approximately 62% in the number of answered requests when compared to FCFS and 23% when compared to Min-Min. This decrease in the number of answered requests with the Dynamic Min-Min algorithm is related to the reduction in the amount of computing resources with lower computational power. According to this algorithm, heavier requests are attributed to resources with lower computational power, which, because they are in smaller quantities in Case 4, generate queues composed of heavy requests.

B. Scenario 2

For the second experiment scenario, three algorithms were used: Dynamic Min-Min, for baseline of comparison between the two scenarios and because it is an algorithm developed in this paper; Ants Colony, because it is an artificial intelligence algorithm; and finally, DDEU, a heuristic also proposed in this paper, which aims to ensure the SLA and the efficient use of resources.

In this scenario, the observation time was twice that used in the previous scenario, 172,800 seconds, equivalent to two consecutive days of simulation. This was used as a basis because the Ant Colony and DDEU heuristics are algorithms that adapt themselves over time and usually require a larger initial time (warm up) to start balancing the load in a more efficient way. In addition, it is worth mentioning that the deadline for executing a request was 1000 milliseconds, with a rate of variation of 15%.

In Figure 3 and Table V, it can be verified that in all cases the Ant Colony algorithm presented the shortest requests execution mean times, in milliseconds. In Case 1 there was a reduction of approximately 47% considering the Dynamic Min-Min algorithm and 28% over the DDEU. In Case 2, this reduction was approximately 22% for both algorithms. For Case 3, the Ant Colony reduced the EMT by approximately 39% when compared to Dynamic Min-Min and 37% when compared to DDEU. Finally, this reduction was approximately 84% and 50%, respectively, in Case 4. However, only in Case 2 the EMT obtained by the Ant Colony heuristic respected the deadline defined in the SLA. Thus, although it presented the shortest times, it was considered inefficient with respect to compliance with the SLA. On the other hand, it was verified that both Dynamic Min-Min and DDEU algorithms kept the EMTs within the defined deadline, respecting the SLA.

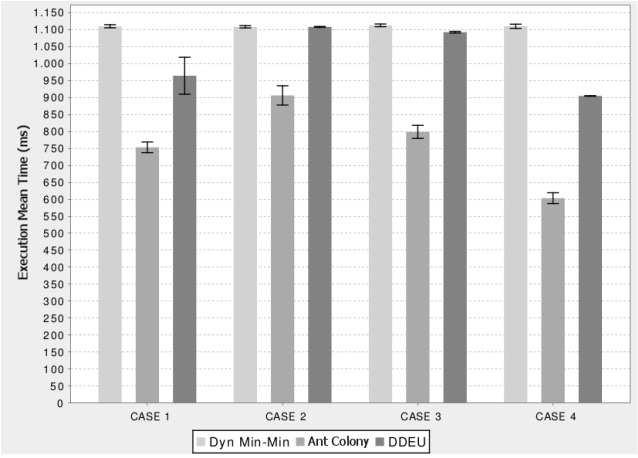

Fig. 3. Response mean times in Scenario 2.

TABLE V
COMPARISON OF THE EXECUTION MEAN TIMES.

Case	Algorithm	EMT (ms)	Standard Deviation	Interval Confidence	Lower Limit	Upper Limit
Case 1	Dynamic Min-Min	1109.69	7.43	4.60	1105.08	1114.29
Case 1	Ant Colony	753.15	25.48	15.79	737.36	768.94
Case 1	DDEU	963.92	87.85	54.45	909.47	1018.36
Case 2	Dynamic Min-Min	1108.06	5.46	3.38	1104.67	1111.44
Case 2	Ant Colony	906.02	45.81	28.39	877.63	934.41
Case 2	DDEU	1107.77	2.27	1.41	1106.36	1109.17
Case 3	Dynamic Min-Min	1112.39	6.48	4.02	1108.37	1116.41
Case 3	Ant Colony	798.62	31.04	19.24	779.38	817.86
Case 3	DDEU	1092.27	3.88	2.41	1089.97	1094.68
Case 4	Dynamic Min-Min	1109.66	10.01	6.20	1003.46	1115.87
Case 4	Ant Colony	603.41	25.77	15.97	587.44	619.38
Case 4	DDEU	904.44	1.28	0.79	903.65	905.23

In terms of the average number of answered requests (Figure 4 and Table VI), it was observed that the DDEU algorithm presented the best results because it performs a more intrusive

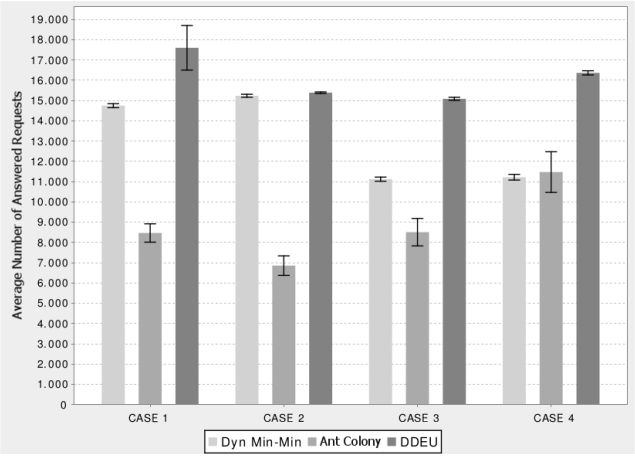

Fig. 4. Average number of answered requests in Scenario 2.

TABLE VI
COMPARISON OF THE AVERAGE NUMBER OF ANSWERED REQUESTS.

Case	Algorithm	AR	Standard Deviation	Interval Confidence	Lower Limit	Upper Limit
Case 1	Dynamic Min-Min	14748.80	158.24	98.08	14650.72	14846.88
Case 1	Ant Colony	8467.30	728.96	451.82	8015.48	8919.12
Case 1	DDEU	17605.70	1772.72	1098.74	16506.96	18704.44
Case 2	Dynamic Min-Min	15238.30	119.03	73.77	15164.53	15312.07
Case 2	Ant Colony	6858.80	773.97	479.71	6379.09	7338.51
Case 2	DDEU	15391.50	60.22	37.32	15354.18	15428.82
Case 3	Dynamic Min-Min	11124.00	164.30	101.84	11022.16	11225.84
Case 3	Ant Colony	8507.80	1094.00	678.07	7829.73	9185.87
Case 3	DDEU	15088.80	120.49	74,68	15014.12	15163.48
Case 4	Dynamic Min-Min	11220.80	226.94	140.66	11080.14	11361.46
Case 4	Ant Colony	11472.80	1618.92	1003.42	10469.38	12476.22
Case 4	DDEU	16368.00	166.44	103.16	16264.84	16471.16

analysis on the computational resources, allowing a fairer requests distribution. In Case 1, the DDEU algorithm presented an increase in the average number of answered requests of approximately 16% and 52% in relation to Dynamic Min-Min and Ant Colony, respectively. For Case 2, with predominance of instances with lower computational power (*m3.medium* and *m3.large*), DDEU proved to be more efficient than the Ant Colony, answering 55% more requests. On the other hand, in relation to Dynamic Min-Min the difference was minimal, only 1%. In the environment with predominance of intermediate instances (Case 3), the percentage of DDEU superiority in relation to Dynamic Min-Min and Ant Colony was approximately 26% and 44%, respectively. Finally, in Case 4, there was an improvement in the average number of requests answered by the Ant Colony algorithm, so that there is no statistical difference between it and the Dynamic Min-Min approach. On the other hand, DDEU provided an increase of approximately 31% in relation to these two algorithms.

It is worth mentioning that, although the Ant Colony algorithm obtained the lowest EMTs, it also obtained the worst performance when the average number of answered requests is considered. This is because this algorithm does not make an efficient use of computational resources. However, as the computational power of the virtual machines increased, there was an increase in the average number of answered requests with this algorithm, in the order of 40% from Case 2 to Case 4.

VI. CONCLUSIONS

Since the users service requests are executed in the virtual machines available in the provider, it is necessary to study and analyze load balancing mechanisms so that the mapping between the requests and the VMs can be more efficient, guaranteeing the QoS defined in the SLA. For this reason, this paper developed and evaluated different load balancing heuristics using different scenarios of experiments through ReMM.

Four distinct configurations of predominance of virtual machine types were defined, which impacted on the provider processing capacity. In this way, it was possible to compare and analyze the behavior of five heuristics, FCFS, Min-Min, Dynamic Min-Min, Ant Colony and DDEU, with variations in processing capacity, exploring the heterogeneity of a cloud environment.

In Scenario 1 results, it was found that the proposed algorithm called Dynamic Min-Min was more efficient than the traditional algorithms FCFS and Min-Min (its predecessor), guaranteeing the defined QoS in the SLA. In Scenario 2, the other proposed algorithm, called DDEU, also presented better throughput in all analyzed cases, since it performs a more intrusive analysis on the requests and on the groups of resources defined by it. In addition, for environments with predominance of more powerful instances, DDEU was more efficient than Dynamic Min-Min algorithm.

Further studies should be conducted that will be based on the results and findings obtained in the course of this paper. They include the development of a prototype using real machines, which will allow a comparison of data from simulated and prototyped environments, as well as analyzes of different network topologies, service demands and computational resources.

ACKNOWLEDGMENTS

The authors thanks FAPESB, FAPEMIG, CAPES, CNPq, UNIFEI and DDMX for the financial support.

REFERENCES

[1] D. Ardagna, G. Casale, M. Ciavotta, J. F. Pérez, and W. Wang. Quality-of-service in cloud computing: modeling techniques and their applications. *Journal of Internet Services and Applications*, 5(1):1–17, 2014.

[2] B. G. Batista, J. C. Estrella, C. H. G. Ferreira, D. M. Leite Filho, L. H. V. Nakamura, S. Reiff-Marganiec, M. J. Santana, and R. H. C. Santana. Performance evaluation of resource management in cloud computing environments. *PloS one*, 10(11):21, 2015.

[3] B. G. Batista, C. H. G. Ferreira, D. C. M. Segura, D. M. Leite Filho, and M. L. M. Peixoto. A qos-driven approach for cloud computing addressing attributes of performance and security. *Future Generation Computer Systems*, 68:260–274, 2017.

[4] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[5] K. Dasgupta, B. Mandal, P. Dutta, J. K. Mandal, and S. Dam. A genetic algorithm (ga) based load balancing strategy for cloud computing. *Procedia Technology*, 10:340–347, 2013.

[6] M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

[7] P. G. Gopinath and S. K. Vasudevan. An in-depth analysis and study of load balancing techniques in the cloud computing environment. *Procedia Computer Science*, 50:427–432, 2015.

[8] R. Jain. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. New York, NY, USA, Wiley, 1991.

[9] P. Mell and T. Grance. The nist definition of cloud computing (draft). *NIST special publication*, 800:145, 2011.

[10] R. R. Selmic, V. V. Phoha, and A. Serwadda. *Quality of Service*. Springer, 2016.

[11] A. Singh, D. Juneja, and M. Malhotra. Autonomous agent based load balancing algorithm in cloud computing. *Procedia Computer Science*, 45:832–841, 2015.

[12] V. Tyagi and T. Kumar. Ort broker policy: Reduce cost and response time using throttled load balancing algorithm. *Procedia Computer Science*, 48:217–221, 2015.

[13] M.-A. Vasile, F. Pop, R.-I. Tutueanu, V. Cristea, and J. Kołodziej. Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing. *Future Generation Computer Systems*, 51:61–71, 2015.

[14] Z. Zhang and X. Zhang. A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation. In *Industrial Mechatronics and Automation (ICIMA), 2010 2nd International Conference on*, volume 2, pages 240–243. IEEE, 2010.