Journal of Theoretical and Applied Information Technology
15th August 2018. Vol.96. No 15

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

ON THE IMPACT OF REAL TIME PARAMETERS INTO THE DESIGN OF CLOUD LOAD BALANCERS

[1]ALI HUSSEIN ALI ALNOOH, [2] DHUHA BASHEER ABDULLAH
[1,2]Department of Computer Science, College of Computer Science and Mathematics,
University of Mosul, Nineveh, Iraq
E-mail: [1]ali.hussein.alnooh@google.com, [2]rtdm_2005@yahoo.com

ABSTRACT

The term "Cloud" has turned out to be universal in our life since it manages a large portion of our activities. Truth be told, cloud benefits are achieved in the vast majority of our social, scholastic, and business tasks. The technological era we are currently living in, is almost based on Internet services. A novel load balancer called Community-Based Cloud Load Balancing Approach (CBCLBA) for distributing tasks to their appropriate server is proposed. The main goal of this paper is to adopt a community based algorithm that takes into considerations tasks time constraints. Our proposed algorithm is implemented on a dataset that is brought from Google. The proposed algorithm shows that when considering two real time parameters, namely; Scheduling Class and Priority, the number of missed deadline tasks decreases comparing to the case when ignoring the two mentioned parameters. Finally, we believe that this is the first kind of work that utilizes concepts from sociology in designing cloud load balancing approaches.

Keywords: *Cloud Load Balancing, Cloud Computing Approaches, Google Clusters-usage traces, Real time*

1. INTRODUCTION

Cloud computing has become a core concept when it comes to cloud services [1]. Most of the companies, universities, and manufactures around the world publish their services to people by targeting what has been called Cloud. Moreover, today's life has witnessed a great revolution in the field of cloud computing [2]. Hundreds (or maybe thousands) of web applications developed and deployed every day to the cloud. These applications are designed either to be for general or for special purposes depending upon market's demands. Nowadays, cloud computing has changed the way of where and how application computations are going to be processed. It has given a lot of attention by the research community and is now considered as a reliable computing model for different application domains. However, developers tend not to use cloud computing for their real time applications. But now, there are many researches in cloud computing field working to develop real time applications and utilize the strength of cloud [3]. In general, application users send their requests as Tasks to a particular cloud service of interest, then it responds back when the request processed. Some of requests (tasks) have time constraints (e.g., deadlines) to be executed before. Therefore, this kind of tasks need to be responded immediately and any latency may lead to missing the deadlines of tasks. This case is not desired because it breaks the reliability feature of the application and eventually makes users unsatisfied [4]. Our Motivation in this work is that the cloud computing literature has a sever lack in algorithms that deal with real time tasks. The need for real time algorithms in cloud computing has become a critical issue since many of the available web services on the cloud need to be responded immediately. For example, cars auctions web sites provide different web services, some of which are considered as real time tasks such as bidding function service. This kind of services (tasks) should be responded immediately at the user bids, while other services such as user settings are not considered as real time tasks (regular tasks).

In this paper, a novel algorithm for real time tasks load balancing is proposed. The proposed method takes into consideration the time constraints of tasks (i.e., scheduling class and priority of tasks). The proposed algorithm also uses concepts from sociology (e.g., social communities and assortativity) in designing its strategies for assigning tasks to servers. This work uses a dataset provided by Google (Google Cluster-Usage Traces) [5]. The ultimate goal of this work is to building a social inspired algorithm for dealing with real time tasks and prevent missing their deadlines as much as possible. This paper is divided into six sections, the next section states the contribution in this paper, and

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

section 3 shows what has been developed in the literature. Section 4 explains our methodology in the proposed method. Section 5 discuss the obtained results and the results of the benchmarking algorithms. Finally, we conclude our work in Section 6.

2. CONTRIBUTION

The main contribution of this paper is designing a novel community-based real time load balancer. The proposed load balancer is inspired from sociology concepts (social community and assortative mixing) and is called Community-Based Cloud Load Balancing Approach (CBCLBA). CBCLBA takes into considerations the time constraints of tasks (e.g., scheduling class and priority) before assigning them to the appropriate servers for execution.

3. RELATED WORKS

Cloud computing developers try to develop algorithms for load balancing that satisfy users' needs and meet application's goals in terms of reliability. Some of the works try to fix the issue of real time during tasks scheduling, while other works try to balance the distribution (load) to servers taking into considerations time constraints of tasks [6]. Although the severe lack and the gaps existed in the real-time cloud load balancing literature, a few researchers worked on critical issues that are related to cloud load balancing. Lee et al., in [7] proposed an algorithm that improved what has been called VMM scheduler to be convenient for soft real time system applications. The algorithm they developed tried to overcome the issue of delay when assigning real time tasks. In their proposed method, the authors used a strategy that is based on allocating the required resources for tasks at the time they need. Belgaum et al. [8] proposed an algorithm for dealing with real time scheduling for services. Their algorithm has the ability to rank services based on the usage-history of these services. Their algorithm also enables users to immediately (real time) access the higher ranked services by communicating with the past users who accessed that services. Another work that deals with the real time issue was performed by Wu et. al. [9] who suggested a real time load balancer for mitigating the load on particular servers.

4. METHODOLOGY

4.1 The Proposed Algorithm

The goal of this paper is to propose a novel community based load balancer for distributing tasks to servers in an unbiased and even way. The proposed load balancer is called Community Based Cloud Load Balancing Approach (CBCLBA) because its design uses social-community concepts. This paper uses Google cluster-usage traces dataset brought from Google [5]. Basically, a Google cluster is a set of servers that are connected to each other. Google clusters share a common management system that assign tasks to machines. Loads arrive at clusters in the form of jobs. Each job includes one task or more, each of which is accompanied by a set of resource requirements used for assigning tasks onto the convenient servers. Also, each task in turn consists of multiple processes; to be executed on a single server.

In this work, we consider n clusters C where Cn represents a vector of clusters $C1, C2, ..., Cn$. We also consider m Servers S where $Cn[Sm]$ represents clusters servers $C1[S1], C1[S2], ..., Cn[Sm]$. Each task has its own requirements to be executed and each server has its own specs (e.g., capacity) for executing tasks. On the other hand, we use the concept of assortative mixing [10] from sociology, which is of two social actors (e.g., two individuals) represents their tendency to attach to each other. Regarding to tasks, we consider information such as resources requirements (e.g., CPU and memory), scheduling class, and priority as the norms and values of tasks. We also consider servers information such as server capacity in terms of CPU, memory and other parameters (e.g., the probability of machines failure in a particular cluster) as the norms and values of servers. We also take the degree centrality [11] into our considerations for servers; this community measurement reflects the number of tasks that are currently loaded on a particular server.

In our proposed algorithm, tasks and servers will be formed as a complex network that can be represented as a graph, of which the vertices (Nodes) V are the tasks and servers where V_l represents a vector of vertices $V1, V2 , ..., V_l$, while the edges E are the relations (ties) among tasks and servers where E_m represents all the edges among network nodes $E_1 , E_2 , ..., E_m$. All the edges among tasks and servers can be determined and formed according to the decision of the proposed load balancer.

CBCLBA has a queue for all the incoming tasks aiming at later assigning each task to its convenient server for execution. When tasks arrive to CBCLBA, it considers that every task has a weak relation (tie) to every single server in the available clusters. CBCLBA, then, calculates the strength of the relation between each pair (task and server). After calculating ties strength among tasks and servers, some ties may become strong. The strength of a tie between a task and a server depends on the values of

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

task Status and Server Status. When a tie between a task and a server become strong, CBCLBA calculates the assortativity level for each strong tie (between a task and a server). The assortativity level will determine whether this task is assigned to a particular server. The assortativity level cannot be calculated if the relation between a task and a server is weak. In CBCLBA, it is possible for a task to have strong ties with many servers but based on the assortativity level, this task will be assigned to one server that will execute that task.

4.1.1 Calculating the Status of Tasks

In CBCLBA, to calculate tasks status we consider T_l is a task that came to CBCLBA. Each task has a priority $T_l(P)$, amount CPU needed for processing $T_l(C_{Needed})$, amount of memory needed $T_l(M_{Needed})$, and scheduling class $T_l(SchC)$. These parameters are available in the dataset we used in this work. We also planned to calculate the deadlines of tasks using some parameters available in the dataset used.

4.1.2 Calculating Tasks Deadlines

Regarding to tasks information in the used dataset, there are two parameters available, namely, *Start Time* ($T_l st$) *and End Time* ($T_l et$). These two parameters can be used in finding the deadlines of tasks. Before that, we should find the execution time of each task cTl , which can be calculated as follows:

$$C_{Tl} = T_l et - T_l st$$

The deadline of a task $\delta(T_l)$ represents the maximum response time for a particular task. Task deadline is important to be considered when calculating the status of a particular task since we are dealing with real time tasks. The deadlines of tasks are not included in the dataset used. Therefore, we propose to calculate tasks deadlines based on the information provided by the dataset as follows:

$$\delta(T_l) = C_{Tl} + C_{Tl} / T_l(SchC) \quad (1)$$

Where C_{Tl} denotes the execution time of the task T_l and $T_l(SchC)$ is the scheduling class of that task. We form this equation to be appropriate with the dataset parameters since it makes sense when including the scheduling class of a task and express the latency sensitive concept that was used in the dataset. According to Equation 1, the high the value of sensitivity to latency, the close the deadline is. This fact is reflected in our proposed equation for deadline calculations. Furthermore, Equation 1 is also used for benchmarking purposes.

It should be mentioned that when a task assigned and migrated to its server, CBCLBA will recalculate the deadlines of tasks. Each task at CBCLBA has to spend an amount of time (processing time) until it is assigned to a particular server. The processing time of tasks ptTl in CBCLBA will be further subtracted from the deadline calculated in Equation 1 and used later by the assigned server for tasks to meet their deadlines as follows:

$$\delta(T_l) = cT_l + (cT_l / T_l(SchC)) - ptT_l \quad (2)$$

This means that Equation 1 will be used in CBCLBA, while Equation 2 is used after the assignment. This, actually, is reasonable because each task at CBCLBA consume time during the assigning process, which must be taken into considerations before migrating tasks to their servers.

Based on the aforementioned parameters, we calculate the status values of tasks at CBCLBA, in which each task has its own status value; this value represents the status of this task in the community of tasks. A task status $\Psi(T_l)$ is the collective value of the resource requirements and the urgently of a task and can be calculated as follows:

$$\Psi(T_l) = (\rho(T_l) + \upsilon(T_l)) \quad (3)$$

Where $\rho(T_l)$ represents the resource requirements of task T_l , which is:

$$\rho(T_l) = T_l(C_{Needed}) + T_l(M_{Needed}) \quad (4)$$

and $\upsilon(T_l)$ represents the urgently of executing a task (or how urgent a task to be executed) and can be calculated as follows:

$$\upsilon(T_l) = T_l(P) + T_l(SchC) \quad (5)$$

Now, Algorithm 1 shows the steps of calculating tasks status at CBCLBA.

Algorithm 1 Steps for calculating tasks status at CBCLBA

INPUTS: CPU and memory requirements, priorities, and scheduling class of tasks
for all *Task* $\in (T_l)$ at CBCLBA do
Calculate deadlines using Equation 1
Calculate task status using Equation 3
end for

For the sake of the parameters and the experiments to be performed in a more convenient way in terms of the variety in the values of parameter, all the parameters have been normalized.

4.1.3 Calculating the Status of Servers

In the same way of calculating the status of tasks at CBCLBA, we calculate the status of cluster servers. As mentioned before, a cluster has many

Journal of Theoretical and Applied Information Technology
15th August 2018. Vol.96. No 15

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

servers and each server has its own specs. Each server *Cn[Sm]* has a particular capacity of CPU *Cn[SmC]* and memory *Cn[SmM]*. Moreover, each cluster has a different probability of having a Server Failure SF than the servers in other clusters. Therefore, the probability of a server failure at a cluster *PrSF(Cn)* is fixed within the same cluster and is varied among different clusters. Each server within a cluster has a number of tasks assigned to it (load). The number of these connections plays a significant role in consuming server resources. In the context of complex networks and social communities, these connections represent the degree centrality of a server *d(Cn[Sm])*. Therefore, it is important to consider this issue at CBCLBA. To avoid having unbalanced servers load, we consider the degree centrality of each server when calculating server status. The status of a server *Ψ(Cn[Sm])* in a particular cluster can be calculated as follows:

$$\psi(Cn[Sm]) = (R(Cn[Sm])) - (\Pr SF(Cn))^{(d(Cn[Sm]))} \quad (6)$$

Where *R(Cn[Sm])* represents the collective values of the server *Cn[Sm]* CPU and memory capacities as follows:

$$R(Cn\ [Sm\ ]) = Cn\ [Sm\ C] + Cn\ [Sm\ M] \quad (7)$$

The term *d(Cn[Sm])* will significantly contribute in distinguishing the servers that have heavy load than the other servers with light load. This consideration is fair enough for servers and the proposed load balancer. Algorithm 2 shows the steps for calculating the status of servers**.**

Algorithm 2 Steps for calculating servers status at CBCLBA

INPUTS: Servers capacities in terms of CPU and memory
for all *Server* $\in Cn[Sm]$ defined at CBCLBA **do**
Calculate the degree centrality (Server Load)
Estimate the probability of failure at cluster Cn
Calculate server Status using Equation 6
end for

4.2 Assigning Tasks to Servers

After calculating the status of tasks and servers by CBCLBA, the next procedure is performing the assignment process. Before we explain the technique we used in the assigning process, many facts on our work should be described and explained. As mentioned, the dataset we used consists of tasks and servers. This combination formed a complex network, of which the nodes are the tasks and servers, and the edges represent the relations among nodes (task and servers). According to the pre-experiments performed on part of the dataset, the degree distribution of this network reflects a power law distribution (long-tail distribution) [12] as shown in Figure 1., which is expected because the number of tasks executed (or assigned) are significantly much greater than the number of servers. Since the degree distribution of our CBCLBA network follows a power-law distribution, it can be said that our network is considered to be a scale-free network according to [13] and [14].

In scale-free networks, the concept of preferential attachment [15] plays an important role in attaching nodes to each other.

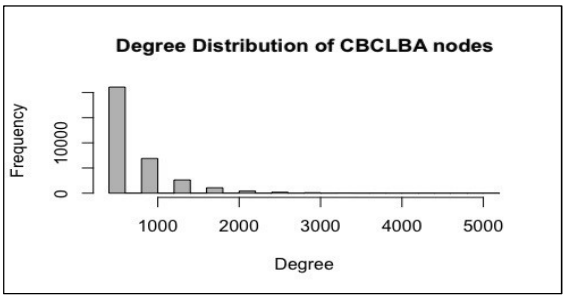

Figure 1: Degree Distribution of CBCLBA Nodes.

In the context of our work, the concept of preferential attachment can be expressed by the assortitivity level of two nodes. CBCLBA will take into consideration the assortativity level between a task and a server when assigning tasks to servers. The question now is how CBCLBA assigns a task with a status of *Ψ(Tl)* to a server with a status of *Ψ(Cn[Sm])*?. To answer this question, the values of status for tasks and servers should be studied and analyzed in a way that helps CBCLBA to make the convenient decision.

Now, according to the preliminary experiments performed (taking different samples of data values for tasks status), we found that the distribution of status values follows a power-law distribution as shown in Figure 2.

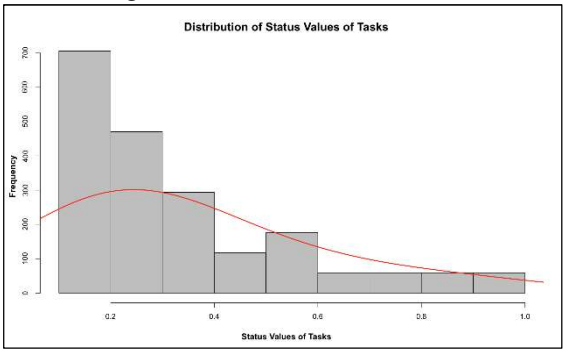

Figure 2: Distribution of Tasks Status.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

This, in fact, is very interesting finding since it enables us to classify status values to classes (or communities) [16]. Moreover, this finding confirms the fact that most of tasks are regular tasks with low or medium scheduling class and with low or medium priorities.

Based on the characteristics of power-law distribution, it is possible to apply Pareto Rule (or called 80/20 rule) on our tasks status data [17]. This rule is applicable when the distribution of the data used follows a power-law distribution. Pareto rule states that about 80% of the events are caused by the other 20% of the population. Figure 3 depicts Pareto rule (80/20 rule).

Our idea proposes that Pareto rule makes it possible to classify status data into three classes or three communities (i.e., CBCLBA is clustered into three different communities, each of which has its own characteristics and each node within any community

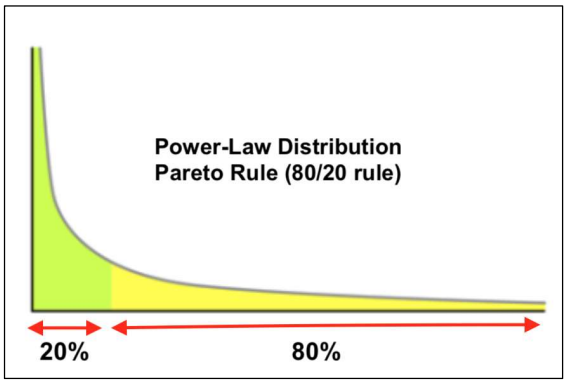

Figure 3: Description of Pareto Rule.

has relatively similar features.), as follows:

• **Low Level Community Class (LLC):** containing the lowest 20% of the values of tasks status.

• **Medium Level Community Class (MLC):** containing lowest 40% of the remaining 80% of the tasks status.

• **High Level Community Class (HLC):** containing highest 40% of the remaining 80% of the tasks status.

As expected the status values of servers reflect roughly close results to tasks as shown in Figure 4. The distribution of status values of servers also followed a power law distribution. This means that Pareto rule is also applicable when we classify the status values of servers. CBCLBA also will use the same classes that are listed above in classifying servers' status. After classifying the status values of tasks and servers by CBCLBA, now it is possible to assign a task to a particular server.

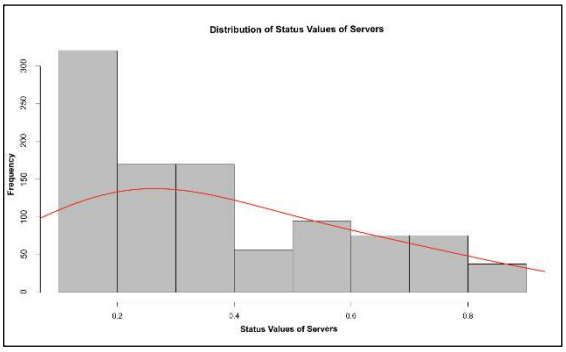

Figure 4: Distribution of Server Status.

In CBCLBA, we propose that tasks and servers that belong to the same community class of status will have strong ties among them. In this case, it is possible for a task to have many strong ties with other servers under the same cluster and there exists many servers available as candidates to serve this task.

4.3 Proactive Assignment Strategy (PAS)

The strategy we propose in assigning tasks to servers called Proactive Assignment Strategy (PAS), this strategy states that a task with a status value of $\Psi(T_l)$ is assigned to a particular server with a status value of $\Psi(Cn[Sm])$ if and only if the assortativity level between both is maximum comparing to the levels with all the servers that have strong ties with that task. In this work, a high value of assortativity level (close to 1) between two nodes reflects their strong tendency to connect and attach to each other. Furthermore, the candidate server should not be at its maximum load when it is selected for a particular task. In addition to the server assigned to a task, we propose that a task should have a second option (backup server) in case of original server failure occurs (e.g., fault tolerance). Yet, PAS strategy skips the backup server and re-perform the assignment process under one condition; if the processing time of the previous assignment process was relatively small comparing to its execution time. Practically, we consider this step when the time of the previous assignment process of the task is in the range of [≈ 0.005%, ≈ 0.01%] from the total execution time of that task. This percentage came from the preliminary experiments we performed for part of the dataset used. Also, under this percentage, tasks are considered safe in terms of meeting their deadline since this amount is very small comparing to its total execution time and deadline. Finally, PAS strategy can be summarized in three cases as follows:

• **Case I (the use of original server):** It is the first option for a task to be assigned. The original server represents the most appropriate server among all the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

server that a task has strong ties with. A task will definitely be assigned to this server if there is no server failure occurs.

• **Case II (the use of backup server):** Each task at CBCLBA has a backup server and is considered the second option for a task when server failure occurs on the original server. It is no guarantee that a task will be assigned to it because a condition must be satisfied before assigning tasks to their backup servers (see Case III).

• **Case III (re-assigning tasks):** As mentioned before, in case of the processing time is significantly smaller than the execution time of the task, CBCLBA will ignore the backup server and re-perform the assigning process again.

4.4 Calculating The Assortativity Level

As mentioned, in this work we propose to use the concept of assortativity between two nodes when assigning tasks to servers. The assortativity level of a pair of nodes (task and server) can be calculated based on the *Assortativity Coefficient r* [10] of the task and servers that have strong ties among them. The assortativity level is often calculated using the degrees (number of connections) of a pair of nodes. However, in this work we propose at using the status of tasks and servers in calculating the assortativity level between two nodes and can be as follows:

Lets assume that j denotes status of the task $\Psi(T_l)$ and k denotes the status of server $\Psi(Cn[Sm])$. The Assortativity level between the status of the pair (T_l , $Cn[Sm]$) can be expressed by the Assortativity coefficient r as follows:

$$r_{jk} = \frac{\sum_{jk} jk(e_{jk} - q_j q_k)}{\sigma_q^2} \qquad (8)$$

Where q_j , q_k are the complementary status of the task $\Psi(T_l)$ and the server $\Psi(Cn[Sm])$ respectively. σ_q^2 is the variance of a server status $\Psi(Cn[Sm])$ that task T_l attempts to connect to, and can be calculated as follows:

$$\sigma_q^2 = \sum_k k^2 q_k - \left[\sum_k k q_k\right]^2 \qquad (9)$$

The term q_k is the remaining status distribution of a server. The distribution of q_k is derived from the total network server's status distribution p_k as q_k can be calculated as:

$$q_k = \frac{(k+1)p_{k+1}}{\sum_j j p_j} \qquad (10)$$

Algorithm 3 describes the steps of assigning a task to its corresponding server and calculates the assortativity level between them.

Algorithm 3 Assigning tasks to servers and calculate Assortativity among tasks and servers at CBCLBA

INPUTS: tasks status $\Psi(T_l)$ and servers status $\Psi(Cn[Sm])$
for all $\Psi(T_l)$ and $\Psi(Cn[Sm])$ at CBCLBA do
  Assign each to its corresponding class
  if Class$(\Psi(T_l))$ = Class$(\Psi(Cn[Sm]))$
    Establish a strong tie between T_l and $Cn[Sm]$
  end if
end for
for all T_l has strong ties with $Cn[Sm]$ do
  Calculate the assortativity coefficient r using Equation 8
  Assign
  $T_l \rightarrow$ Server $\in$ Max($r_{T_l,Cn[Sm]},\ldots,r_{T_l,C_l..n[S_{j..m}]}$)
end for

5. EXPRIMENTAL RESULTS

5.1 Benchmarking Methods

This section describes the methods we benchmark our proposed method with. In fact, we choose methods that fits the parameters used in the dataset. These methods are described as follows:

• **Least Connection:** This approach assigns tasks to the servers that has less load of tasks comparing to other servers. It does not take into considerations the capacity of the selected server and tasks requirements [18].

• **Round Robin:** This is a well-known approach in the literature, it distributes the incoming tasks to servers in an even way at each round [18].

• **Shaw's Algorithm:** this algorithm is proposed by [19]. It is a combination of the features of two known approaches in the literature; mainly from Active Monitoring Load Balancing (AMLB), and from VM Assign Load Balancing (VM-ALB). This approach solves the issue of ignoring the servers that have been selected in the current round. It assumes that the used servers in the current round can be employed as candidate servers in the next round if they are available.

5.2 Simulation Environment

In this work, we designed a special-purpose simulator for implementing and testing the proposed algorithm (CBCLBA), Shaw's Algorithm, Round

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Robin and Least Connection algorithms. The programming of the simulator is based on the concept of Multi-Agent. We involve the NetLogo modelling, which is Java-based programming. For all the experiments in the simulations, we in involve approximately 127, 400 tasks and about 50 servers. These data is imported from the original dataset from Google.For more accurate results and better evaluation for all the algorithms used in this work, we ran each experiment for four times and then use the average of the experiments we obtained.

5.3 Performance Evaluation

Each of the algorithms used in this work consumes amount of time for assigning tasks to servers. The time consumed depends on the algorithm used, which in turn depends on the calculations that are needed during the assignment process. We believe that tasks consume more time in CBCLBA than the other algorithms used. However, the benchmarking approaches are not optimal and they are for general purposes. For example, if we have real time tasks that need to be assigned and executed by some servers, it is necessarily needed to have a load balancer that takes into account the deadlines of tasks and prevents tasks from missing their deadlines as much as possible.

Each task arrives to the queue of the load balancer and has its own resource requirements. Some of the arrived tasks have deadlines to be executed before. These real time tasks have time constraints that should be taken into considerations by load balancer designers.

In our simulations, we simulated four algorithms, of which one is the proposed algorithm (CBCLBA) and three are the benchmarking algorithms. First, we test all the algorithms in terms of missed deadlines tasks. We include all the tasks we have (127, 400) in the simulations of the algorithms performed in this paper. After the simulations completed, we compare the time of each task after assigning it to a particular server against its deadline regardless the algorithm used. We, then, counted the number of tasks that missed their deadlines. This step will help us in evaluating the proposed algorithm. Figure 5 depicts the number of tasks that missed their deadlines.

The numbers showed in the figure reflect the means of all the runs we performed for each algorithm (each algorithm is performed for 4 times). According to Figure 5, it is clear that CBCLBA outperformed all the other algorithms in terms of the number of missed deadlines tasks, while Round Robin reflects the worst performance since there are many tasks missed their deadlines. Also, Shaw's algorithms reflect better performance than least connection and round robin, but it underperformed CBCLBA.

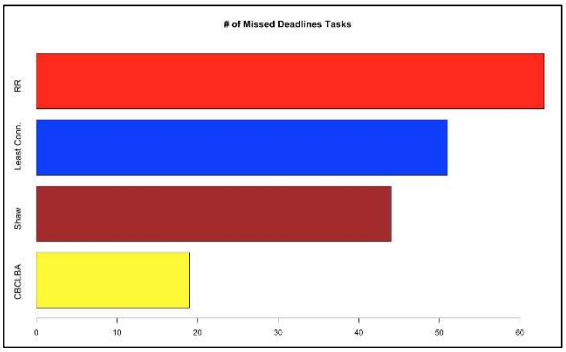

Figure 5: The number of tasks that missed their deadlines during the assignment process.

The findings above are not enough for judging on CBCLBA to be the best and outperformed the other algorithms. Therefore, additional analysis should be performed to confirm that these results are statistically significant comparing to the benchmarking methods. To this end, it is needed to test and analyze the variations of all the algorithms and find which algorithm has less variations. We start with the boxplot of the variations of the algorithms. Figure 6 shows the detailed variations in terms of median and the quartiles.

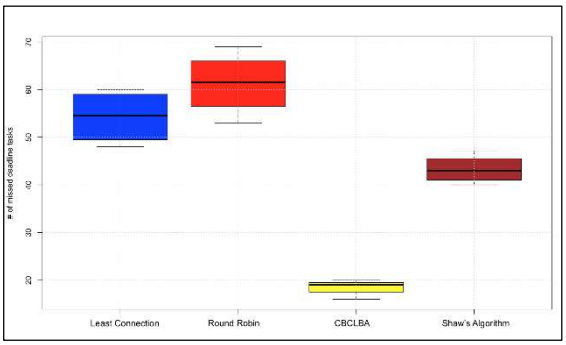

Figure 6: A boxplot that shows the variations of each algorithm used in terms of the numbers of missed deadlines tasks.

This boxplot also shows that there are no outliers appeared, which is considered as a positive point for all the results. This figure also reflects the stable behavior of CBCLBA in terms of results variations, while the other algorithms reflect relatively wide range of variations (see Table 1). This means that the stability level of the proposed algorithm is higher than the other algorithms.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1: The variations of algorithms in terms of the number of missed deadlines tasks.

Algorithm Performed	Variations Range	Stability Level ST_{Level} (Low, Medium, and High)
CBCLBA	16 – 20	High
Shaw	39 – 47	Medium
Least Conn.	48 – 60	Low
Round Robin	53 – 69	Low

Now, it is needed to confirm these results by using the One-Way ANOVA technique for the variations of each algorithm. It is also needed to test whether the mean values μ of the variations for each algorithms. To do so, we have to assume and perform two hypothesis testing. The first hypothesis assumes that the mean values of all the algorithms are similar and the second (alternative) hypothesis assumes not. Below are the forms of our hypothesis:

Null Hypothesis:
H0 : ∃ CBCLBA, Shaw, L Conn, RR: $\mu C\ BC\ LBA = \mu Shaw = \mu LC\ onn = \mu RR$.
Alternative Hypothesis:
Ha : ∃ CBCLBA, Shaw, L Conn, RR: $\mu C\ BC\ LBA = \mu Shaw = \mu LC\ onn = \mu RR$.

The results of ANOVA analysis shows that *F-statistics* is 63.52 and *P-Value* is 1.24e − 07, this means that the value of *P* is significantly smaller than the value of *F-Statistics*. Therefore, we reject the Null hypothesis of similar means of the algorithms and accept the Alternative hypothesis. These findings confirm all the aforementioned results we discussed before. Table 2 shows the output of ANOVA for the used algorithms in this work.

Table 2: One-Way ANOVA table for the variations of the algorithms used in terms of missed deadlines tasks.

	SumSq	MeanSq	F-Stat.	P-Value
Effect	4212	1404.1	63.52	1.24e-07
Residuals	265	22.1		

Now, the final step of this analysis is to calculate and show the variations among all the algorithms. The goal of this step is to compare the variations of each pair of algorithms for obtaining accurate evaluation of the algorithms. To this end, we use *Bonferroni* test method for calculating the matrix of variations among all the algorithms. This technique is considered as a pairwise comparisons using t tests with pooled SD. Table 3 shows the values calculated for each pair of the used algorithms.

Table 3: Bonferroni Test for pair-wise comparison for the missed deadlines variations of the proposed and benchmarking algorithms.

	Least Conn.	Round Robin	Shaw
Round Robin	0.34185	-	-
Shaw	0.03743	0.00094	-
CBLBA	9.7e − 07	1.3e − 07	4.7e − 05

In this table, it can be observed that CBCLBA outperformed Least Connection, Round Robin, and Shaw's algorithms with negative pair values of 9.7e − 07, 1.3e − 07, and 4.7e − 05 respectively.
Furthermore, Shaw algorithm reflects a close behaviour in terms of variations to Least Connection, and Round Robin with positive values of 0.03743 and 0.00094 respectively. The behaviour of the proposed algorithm seems to be more stable and has less variations the the benchmarking algorithms.

Finally, the performance of each algorithms can be summarised by Table 4. It shows the percentage (P_{total}) of missed deadlines tasks to the total number of tasks used, the stability level ST_{Level} , variations level Var_{Level} , and real time level RT_{Level}.

Table 4: Summarizing the performance of the algorithms used.

Algorithm	P_{total}	ST_{total}	Var_{Level}	RT_{Level}
CBCLBA	1.5%	High	Low	Real Time
Shaw	3.6%	Med.	Med.	No Real Time
Least Conn.	4.7%	Low	High	No Real Time
Round Robin	5.4%	Low	High	No Real Time

According to the aforementioned results and in spite of the reliable performance of the proposed

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

algorithm, it has some pros and cons in terms of level of complexity and processing time. The proposed algorithm is more complex to implement than the other algorithms mentioned in this work. The processing time that each task spend in the proposed load balancer is more than the time consumed using the benchmarked algorithms, this is due to the former reason. The weaknesses points of the proposed algorithm are planned to be avoided and addressees in our future works since this is an ongoing work.

6. CONCLUSION AND FUTURE WORKS

In this paper, we propose a novel algorithm for designing a community-based real time load balancer. Our algorithm (CBCLBA) is inspired from sociological concepts among people such as social communities, assortative mixing, and degree centrality. We also involved time constraints for tasks in the design of CBCLBA such as tasks priority, tasks scheduling class, tasks execution times, and tasks deadlines. The dataset we use in this work is brought from Google (cluster-usage traces), which contains real tasks. We benchmark the proposed CBCLBA with three known algorithms in the literature, namely, Round Robin, Least Connection, and Shaw's algorithm for cloud load balancing. According to the experimental results, CBCLBA outperformed the benchmarking algorithms in terms of the number of missed deadline tasks and in terms of performance variations. We also confirmed all the results and proved that the obtained performance is statistically significant comparing to the benchmarking algorithms. Moreover, it is clear that using concepts from sociology such as the characteristics of social communities, assortative mixing, and degree centrality are considered as powerful tools in designing effective load balancing algorithms. As a future work, we plan to develop CBCLBA and add more real time related parameters to reduce the number of missed deadline tasks and improve the performance of the proposed method in terms of time constraints.

REFRENCES:

[1] P. Mell, T. Grance et al., "The NIST definition of cloud computing," 2011.

[2] D. C. Marinescu, Cloud computing: *theory and practice*. Morgan Kaufmann, 2017.

[3] S. Malik and F. Huet, "Adaptive fault tolerance in real time cloud computing," in *Services (SERVICES), 2011 IEEE World Congress on*. IEEE, 2011, pp. 280–287.

[4] J. Zhao, Y. Xiang, T. Lan, H. H. Huang, and S. Subramaniam, "Elastic reliability optimization through peer-to-peer checkpointing in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 2, pp. 491–502, 2017.

[5] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format+schema," *Google Inc.*, *White Paper*, pp. 1–14, 2011.

[6] S. Sheikh and A. Nagaraju, "A comparative study of task scheduling and load balancing techniques with mct using etc on computational grids," *Indian Journal of Science and Technology*, vol. 10, no. 32, 2017.

[7] M. Lee, A. S. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, "Supporting soft real time tasks in the xen hypervisor," *in Proceedings of the 6th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE '10. New York, NY, USA: ACM, 2010, pp. 97–108. [Online]. Available: http://doi.acm.org/10.1145/1735997.1736012

[8] M. R. Belgaum, S. Soomro, Z. Alansari, and M. Alam, "Cloud service ranking using checkpoint-based load balancing in real time scheduling of cloud computing," in *Progress in Advanced Computing and Intelligent Engineering*. Springer, 2018, pp. 667–676.

[9] Y. Wu, X. Song, and G. Gong, "Real time load balancing scheduling algorithm for periodic simulation models," *Simulation Modelling Practice and Theory*, vol. 52, pp. 123–134, 2015.

[10] M. E. Newman, "Assortative mixing in networks," *Physical review letters*, vol. 89, no. 20, p. 208701,2002.

[11] Y.-Y. Liu, J.-J. Slotine, and A.-L. Baraba´si, "Control centrality and hierarchical structure in complex networks," *Plos one*, vol. 7, no. 9, p. e44459, 2012.

[12] A. Clauset, C. R. Shalizi, and M. E. Newman, "Power-law distributions in empirical data," *SIAM review*, vol. 51, no. 4, pp. 661–703, 2009.

[13] A.-L. Baraba´si and E. Bonabeau, "Scale-free networks," *Scientific american*, vol. 288, no. 5, pp. 60–69, 2003.

[14] A.-L. Baraba´si, "Scale-free networks: a decade and beyond," *science,* vol. 325, no. 5939, pp. 412–413, 2009.

[15] A.-L. Baraba´si, R. Albert, and H. Jeong, "Mean-field theory for scale-free random networks," *Physica A: Statistical Mechanics and its*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Applications, vol. 272, no. 1-2, pp. 173–187, 1999.

[16] G. Palla, A.-L. Baraba´si, and T. Vicsek, "Quantifying social group evolution," *Nature*, vol. 446, no. 7136, p. 664, 2007.

[17] M. E. Newman, "Power laws, pareto distributions and zipf's law," *Contemporary physics*, vol. 46, no. 5, pp. 323–351, 2005.

[18] K. Al Nuaimi, N. Mohamed, M. Al Nuaimi, and J. Al-Jaroodi, "A survey of load balancing in cloud computing: Challenges and algorithms," in *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium* on. IEEE, 2012, pp. 137–142.

[19] S. B. Shaw, "Balancing load of cloud data center using efficient task scheduling algorithm," *International Journal of Computer Applications*, vol. 159, no. 5, pp. 1–5, 2017.