Received: September 9, 2023. Revised: November 4, 2023.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Dynamic Allocation of Weights Using the Minimally Connected Method to Augment Load Equilibrium in Distributed Systems

Mahdi S. Almhanna[1]* **Tariq A. Murshedi[1]** **Ahmed M. Al-Salih[1]** **Rafah M. Almuttairi[2]**

[1]*Department of Information Networks, College of Information Technology, University of Babylon, Babylon, Iraq*
[2]*Collage of Information Technology Engineering, Al-Zahraa University for Women, 56001, Karbala, Iraq*
Corresponding author' Email: mahdi.almhanna@uobabylon.edu.iq

Abstract: Load balancing is critical to managing server resources efficiently and ensuring optimal performance in distributed systems. The weighted round robin (WRR) algorithm is commonly used to allocate incoming requests among servers based on their assigned weights. However, static weights may not reflect the changing demands of servers, leading to imbalanced workloads. To address this issue, this study proposes a dynamic mechanism for assigning weights to servers in the WRR algorithm based on the data rate and incorporates the least connection approach for the best result. The dynamic mechanism considers each server's real-time data rate, representing its current load. Servers with higher data rates are assigned higher weights to attract a larger share of incoming requests, while those with lower data rates receive lower weights to manage their loads effectively. This dynamic weight assignment allows the algorithm to adapt to varying workloads and achieve better load balancing across servers. To further refine the distribution of requests, the least connection approach is employed to handle tie-breaking situations and for more fairness in distributing the loads. The proposed algorithm is a hybrid of data rate and the least connection, it is evaluated through simulations and real-world experiments. The results demonstrate its superiority in achieving improved load balance compared to other algorithms, such as round-robin RR and traditional static-weight WRR algorithms. By dynamically adjusting weights based on data rate and employing the least connection approach, the algorithm optimizes server resource usage, minimizes response times, and enhances overall system performance in distributed environments.

Keywords: Distributed system, Cloud computing, Grid computing, Load balancing, Weighted round-robin algorithm.

1. Introduction

Data network management has become increasingly important due to the growing demand for services related to big data applications. Traditional routing strategies, while effective in the past, have now become prohibitively expensive to implement and maintain [1]. Additionally, with the rise of distributed data storage that generates vast amounts of data, the process has become time-consuming and burdensome.

To address these challenges, a robust load balancer is a viable solution for efficiently managing big data in such data centers.

Various methodologies have been proposed to optimize data network management by leveraging load-balancing techniques and capitalizing on the inherent advantages of the data network [2]. Among the standard routing policies offered by a grid load balancer, the weighted round-robin stands out [3]. However, for this policy to be truly effective, determining appropriate weights for each server in the system is crucial. To address this concern, we propose to employ dynamic programming methods that can intelligently distribute the workload among servers, considering specific characteristics that influence the volume of data to be processed.

The primary objective of our research is to devise an efficient strategy that assigns appropriate weight factors to each server within the system, thereby ensuring a balanced distribution of heavy data loads and optimizing resource utilization at every location. By achieving a fair distribution of data and workload,

Received: September 9, 2023. Revised: November 4, 2023.

our proposed approach seeks to maximize the benefits derived from the available resources, providing a more sustainable and effective data network management solution.

Load balancer helps servers transfer data efficiently, improves application delivery resource utilization, and prevents server overload [4]. several load-balancing algorithms differ from each other in terms of simplicity and complexity, it can be mentioned some in general.

In a data grid environment, the schemes of dynamic load balancing of storage management are very effective in system performance during assigning work to available servers at run time [5]. The literature proposed several ideas to improve distributed load-balancing schemes. Considered the central processing unit CPU, main memory, or both CPU and memory [6]. The aim of enhancing policies of load balancing schemes in distributed sites and the data grids is to increase resource utilization [7]. In the current work, a developed load-balancing algorithm is proposed to improve the performance of the system.

One common routing policy in load balancing is a weighted round-robin algorithm used in cloud and grid computing load balancing [8]. However, effective mechanisms are needed to determine the weights assigned to each server to achieve the best balance of the system.

Most balancing algorithms rely on a single feature to decide to distribute a job among servers. For example, the round robin algorithm distributes a load of jobs evenly on servers, regardless of the different features of available servers. Whereas in the weighted round robin algorithm, the difference is, the size of the memory was taken into account. Based on the current traffic characteristics and by using a fixed weight for a weighted round-robin at the start of each base station round the weight of each queue in different categories is dynamically guessed. There are many difficulties in designing a system to control congestion which causes significant delays in data transmission which usually leads results in a mismatch between network resources and acceptable traffic.

Accordingly, it is necessary to adopt all or at least most of the characteristics that will be the main reason for affecting the amount of data transfer in the network, such as bandwidth, bit rate, processing speed, and attempts to use some processors to reduce CPU idle time [9].

Data grids aim to reduce latency and transfer files quickly enough [10]. Load balancing aims to distribute network traffic across several servers. As a result, this will ensure that no single server loads all or most of the requests and therefore may result in the server being unable to meet those requests.

By distributing work fairly (may not evenly), load balancing will improve server responsiveness, reduce idle time, and maximize throughput, moreover, If the client cannot access the backend server through the load balancer, the backend server will be declared unhealthy, it is the efficient and regular distribution of network traffic across multiple servers. As for the load balancer site, it is between the client and the servers, receives incoming requests, and then distributes them to a specific available server so that it can process that request.

Load balancers detect the robustness of servers' resources and do not send tasks to any server unable to fulfill the request. For greater consistency and to keep up with the increasing demands of sustainability, all server resources have to be available and balanced in applications layer 4 or layer 7 of (OSI).

2. Related work

The two primary categories of load distribution techniques are static and dynamic load balancing. Static techniques are used for predictable workloads and do not require prior knowledge of the system state. On the other hand, dynamic techniques are designed for unpredictable workloads and consider the current system state before distributing loads.

Adekunbi A. Adewojo and Julian M. Bass [11], proposed study addresses challenges associated with conventional load balancing methods by integrating crucial server and cloud resource metrics into the algorithm. The identified metrics include CPU utilization, memory utilization, network bandwidth, number of threads running, and network buffers, chosen for their influence on real-time server behavior. Leveraging these server metrics within a load-balancing algorithm allows for the efficient distribution of load according to a server's present capacity, resulting in optimized resource utilization. Each VM is assigned a weight based on its current utilization and capacity, allowing for dynamic adjustments to the probability of utilizing a VM during runtime as its state is evaluated.

Chen and team [12] proposed an architecture and algorithm for dynamic load balancing in cloud services, specifically targeting the uneven distribution of workload on servers. Their approach takes into account both the processing power of servers and their current load, leading to improved response times for digitally load-balanced applications in the cloud.

Ahmed Mohammed and team [13] introduced different scheduling algorithms that are evaluated for quality of service (QoS) in terms of throughput, end-to-end delay, and queuing metrics using a network simulator with 50 mobile nodes.

Shafiq and colleagues [14] introduced an algorithm for dynamic load balancing, focusing on optimizing the allocation of resources and balancing the load of virtual machines (VMs) in the infrastructure as a service (IaaS) cloud model. Their algorithm gives priority to VMs, quality of service (QoS) task parameters, and resource allocation, resulting in a significant enhancement in resource utilization compared to existing dynamic load balancing algorithms.

Preecha Somwang [15] introduces an efficient load-balancing technique using HA proxy in cloud computing, focusing on workload distribution and resource sharing. Round-robin scheduling optimizes cloud storage management, leading to effective load balancing and dynamic replication. Evaluation based on requests per second and failed requests demonstrates a 1,000 request / 6.31-sec performance improvement in cloud computing with fewer false alarms.

In another study, Cruz and colleagues [16] presented the EagerMap algorithm, designed to optimize task mappings and mitigate the issue of a single point of failure in load balancing

A notable limitation of these studies is their oversight regarding the current server load, a crucial factor in load balancing. Disregarding this factor could potentially result in algorithmic failures, as the workload may not be fairly distributed across servers. Additionally, these studies do not ensure an even distribution of workloads due to the deferral of load request sizing. For instance, if the algorithm assigns weights of 3 and 1 to two servers, and server 1 is already burdened with a load of 3 requests, and if three new requests arrive, the algorithm would direct all three requests to server 1. As a result, server 1 would process a total of 6 requests, leaving server 2 idle without any requests. Moreover, even if the number of existing connections is the same, the first three requests might be smaller while the fourth one could be considerably larger. In this scenario, a server with higher capacity may be assigned a smaller task, while a server with lesser capacity could be allocated a disproportionately large workload.

Drawing upon insights from the referenced research, this study introduces a dynamic load-balancing algorithm that strategically employs specific key server metrics to calculate server weights. The proposed algorithm and architecture effectively address performance degradation stemming from flash crowds, resource failures, and vulnerabilities such as single points of failure. Furthermore, this algorithm harmonizes with auto-scaling mechanisms in cloud and grid data centers, facilitating the achievement of its intended goals. In contrast to prior research that had a narrow focus on server metrics, this study delves into server metrics that have a direct impact on applications deployed in the cloud. Additionally, the algorithm takes into account both the current server state, indicated by the number of connections, and the server's capacity to handle loads. This adaptability enables the algorithm to operate seamlessly in multiple environments.

3. Load balancing schemes

3.1 Round robin

Round robin is a soft mechanism to make sure that every client request is redirected to a different server in a circular motion. A defect of this algorithm is that It does not care how many uploads are already on the server that was previously uploaded by users. and considers that all servers have the same characteristics and capabilities, in addition to paying no attention to the location of those servers or even the size of the data directed to the server, as the nature and size of the data vary from one request to another, there is, therefore, no fairness in the distribution of the load, and as a result, this may cause some servers to fail.

3.2 Weighted round robin

Each server is assigned a weight based on criteria chosen by the site administrator; the most widely used criterion is the ability to process traffic on the server. The more weight, the higher the percentage of client requests that the server receives. It is considered a good method and almost devoid of problems. But this depends on how you choose the right weight for the server, also, this technology requires the ability to guess processor engagement which is not possible to guess in networks due to the difference in packet sizes. [3].

3.3 Least connection method

Although round robin doesn't take into consideration the existing load on the server [17], the least connected method does this assessment; the servers with the least active connection will be selected to send requests. Typically, this method provides a good performance.

One of the disadvantages of this technique is that it does not take into account the connections to the server, for example, server S1 has 10 connections, and server S2 has 12. The next request will be sent to the S1 server because it has fewer connections, the S1 server may have 50 connection capacity while the S2 server connection capacity is 150, making the S1 server more likely to fail.

3.4 Least response time

Response time is the time between sending a request packet and receiving the first packet. In this way, the server with the lowest average latency and the least number of active connections, the opportunity to redirect the request to it will be increasing.

This method cannot be considered a good indicator of the capacity and specifications of the server as many factors may be the reason for not communicating with this server, including the inability to provide extensive services or the presence of congestion in the path connecting the client and this server, or perhaps poor bandwidth between the client and the server at a specific time. Also, the response time varies according to the request, the more complex the application, the greater the response time, in addition to the presence of server components near it or at different remote places.

3.5 Least bandwidth method

With this method, the route with the least amount of traffic will be chosen. Since bandwidth is a limited hosting resource, bandwidth consumption should be maintained. When the available bandwidth is used up in most cases your site will be suspended and the following error “Bandwidth limit exceeded 509” will appear.

One of the disadvantages of this method is that the Internet speed or connection speed depends on the rate of data transfer over a network wire or its devices, it is a measure of the speed of data transfer over a wired or wireless connection, measured in bits per second, the greater the bandwidth, the faster the information is transmitted and received. Bandwidth restrictions affect the data transfer rate. In low-bandwidth systems, there are restrictions on the amount of data that can be transferred, low bandwidth means slow network performance. So, choosing a low bandwidth will affect the speed of sending and receiving data.

In this research, in the first stage, the weighted round-robin method will be used by suggesting a new method on how to choose the right weight for each server. Also, in the second stage, the slightest connection strategies are adopted to ensure more fairness in distributing the jobs on the servers taking into account the number of connections which is already connected with the servers.

The proposed algorithm introduces a dynamic weighting mechanism based on request size, server capacities, and consideration of waiting connections from the last round. It optimizes the distribution of requests by assigning appropriate weights to each server, aiming for a more balanced load distribution.

The main advantages of the proposed algorithm are:

- Dynamic weighting: The algorithm dynamically adjusts server weights based on real-time request sizes and server capacities. This adaptability allows for efficient load distribution by considering the varying workload demands on each server.
- Fair load distribution: By factoring in waiting for connections from the last round, the algorithm strives for a more equitable distribution of the load across servers. It addresses the issue of potential overloading on specific servers and ensures a balanced allocation of requests.
- Optimized resource utilization: The algorithm optimizes resource utilization by carefully distributing requests to servers based on their current workload and capacity. This promotes efficient usage of server resources and enhances overall system performance.
- Improved scalability: The algorithm's flexible approach to load balancing accommodates changes in the network's size and configuration. It can easily adapt to varying numbers of servers, making it a scalable solution for evolving server infrastructures.
- Enhanced responsiveness: By considering real-time data rates and matrix diameters, the algorithm can respond dynamically to shifts in workload demands. This responsiveness ensures that the load balancing remains effective and responsive to fluctuations in request patterns.

Dynamic programming is a powerful algorithmic technique used to solve complex problems by breaking them down into smaller overlapping sub-problems and efficiently solving each sub-problem only once, storing its solution for future reference. The approach is often used in optimization and combinatorial problems where the solution space is vast and contains overlapping sub-problems.

The main idea behind dynamic programming is to avoid redundant calculations and optimize the time complexity [18] of the algorithm by utilizing previously computed solutions. This is achieved through the use of memorization, which involves storing the results of sub-problems in a data structure (like an array or a hash table) so that they can be easily accessed and reused when needed.

The basic steps involved in solving a problem using dynamic programming are as follows:

- Identify the problem's recursive nature: Determine if the problem can be broken down into smaller, overlapping sub-problems. This often involves finding a recursive relationship between the original problem and its sub-problems.
- Define the base cases: Identify the simplest sub-problems that can be solved directly without further decomposition. These base cases provide the termination condition for the recursive calls.
- Formulate the recurrence relation: Express the solution to a given problem in terms of solutions to its sub-problems. This recurrence relation is crucial for implementing dynamic programming efficiently.
- Memorization or bottom-up approach: Implement the algorithm using memorization, where the results of sub-problems are stored and reused to avoid redundant calculations or use a bottom-up approach, starting from the base cases and iteratively building up to the solution of the original problem.

Dynamic programming is widely used in various fields, including computer science, operations research, artificial intelligence, economics, and bioinformatics, to solve problems that exhibit overlapping substructures. It helps reduce the time complexity of algorithms and enables efficient solutions to problems that would otherwise be computationally infeasible using naive approaches.

3.6 Data rate and disk usage

Each network connection has a data rate, which is the amount of data sent over a specified period over the network; it is the speed of data transfer from one device to another. Whereas, bandwidth refers to the ability of a link to send or receive a number in several seconds, measured generally in bits or megabytes per second (Mbps).

Usually, the slowest component inside a computer or server is long-term storage, which includes hard drives, often causing a bottleneck in the computer. Disk bytes per second is the rate at which bytes are moved to or from the disk during write or read operations. This provides information about the speed of the disk system, and how busy it is.

The speed at which a certain amount of data is transferred over a given period is called the data transfer rate. Data rate is not directly equivalent to any single factor, it can be influenced by several factors, including bandwidth, CPU speed, memory speed, congestion, selected path, quality of network equipment, data compression techniques, signal interference, and the distance between the sender and receiver in wireless communication. Thus, the sum of these parameters together can determine the amount of data rate.

Knowing the transfer rate If some files are downloaded online or data transferred from one source to another is very important in giving a perception of two situations, The first refers to the speed of data transfer within the network, where the bandwidth component is the influencing component, and the second refers to the processing speed inside the computer transmitting data, where the disk usage, CPU processing speed, and response time are

Table 1. Proposed algorithm vs. RR and WRR

Aspect	**Round Robin (RR)**	**Weighted Round Robin (WRR)**	**Proposed Algorithm**
Load Balancing Approach	Sequential	Based on weights	Dynamic weighting based on CPU utilization, request size, server capacities, and waiting connections
Considers Server Weights	No	Yes	Yes
Considers Waiting for Connections from the Last Round	No	No	Yes (Optimizes distribution considering waiting for connections)
Even Load Distribution	Not guaranteed	Depending on weights	Strives for even distribution based on weights, request sizes, and waiting connections
Potential for Overloading a Server	Yes	Yes	Attempts to minimize overloading by considering various factors

Received: September 9, 2023. Revised: November 4, 2023.

Table 2. Symbols utilized in this article

R	Number of servers
n	number of different tasks
p	total return
CPU	Central Processing Unit
MB/S	Megabyte per second
S	server
m	number of servers
N	index within the range 1 to (n - (i + j) + 1).
CC	current connections
NRs	New requests per server
NC	New connections
SW	Server weight
TC	Total connection
TNR	Total new request
TNC	Total new connections
TW	Total weights
SW	Servers weights
CPW	Connections per weight

influential factors, in both cases, it is directly proportional to the transmission speed between two sources.

3.7 CPU utilization

CPU utilization or CPU usage refers to the amount of work that the CPU does. Actual CPU usage varies according to the amount and type of managed computing tasks. Some tasks require longer CPU time, while others require less time due to resource requirements other than CPU.

The following Table 1 compares the round robin (RR) algorithm, the weighted round robin (WRR) algorithm, and the proposed algorithm, considering various aspects.

Table 2 below illustrates the symbols utilized in this article.

3. Formulation of dynamic programming problems

Suppose we have R servers that will be distributed among n number of different tasks. The yield P depends on the tasks and the amounts of resources allocated to them and the goal is to maximize the total return.

Pi (Ri) denotes the return from the task “i” with the resource Ri then the total return is the same as

$$P(R1, R2 \ldots RN) = P1(R1) + P2(R2) + PN(Rn) \ldots \quad (1)$$

$$R = R1 + R2 + R3 + \ldots\ldots Rn \quad (2)$$

$$Ri >= 0 \text{ and } I = 1,2,3, \ldots\ldots, n \quad (3)$$

The problem is to maximize the total return given by Eq. (1), subject to the constraint Eq. (3),

$$\text{If } fn(R) = \text{MAX}$$
$$0 <= Ri <= R\ [P(R1, R2, R3, \ldots Rn)] = \text{Max}$$
$$[P1(R1) + P2(R2) + P3(R3) + \ldots Pn(Rn)] \ldots \quad (4)$$

Then fn (R) is the maximum return from the distributed R to the n tasks If Rn is the quantity of resource allocated to the nth task such that 0<=Rn<=R, Regardless of the values of Rn, a quantity (R-Rn) of the resource will be distributed amongst (n-1) tasks.

Let fn-1 (R-Rn) denote the return from the (n-1) tasks. Then the total return from the total tasks will be: Pn (Rn)+ f n-1 (R-Rn)

So the optimal choice of Rn will be the maximum of the above function and thus the fundamental dynamic programming model may be expressed as:

$$Fn(R) = \text{Max}\ [\ Pn(Rn) + f\, n\text{-}1(R\text{-}Rn),\ n=2,3. \quad (5)$$

Where f1(R), when n= 1 is obtained from (3) as

$$F1(R) = P1(R) \quad (6)$$

Eq. (5) gives the return from the first task when the whole of resource R is allotted to it.

4. Proposed idea

The amount of work that will be assigned to a server depends on that server's ability to process and the amount of data that can reach it [19]. For example, a server can handle 2 megabytes per second, but the amount of data flowing to it does not exceed one megabyte per second, this means that the server will

Received: September 9, 2023. Revised: November 4, 2023.

Table 3. Performance

DISK MB/S	CPU utilization %
0.4	15.2
0.7	22.2
0.8	38.5
0.9	71.4
1	78.5
1,2	85.6
1.3	92

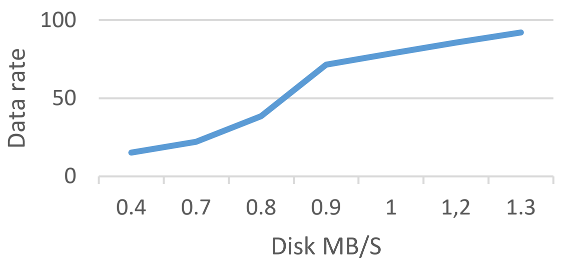

Figure. 1 Relations between data and CPU utilization

operate at half its capacity, On the other hand, it cannot handle a data stream of more than 2 megabytes as a maximum. This means that if we know the server's ability to process, we can limit the amount of data flowing to it to obtain the highest productivity.

Typically, the CPU runs about 30 to 40 percent during non-peak hours, during peak hours, the CPU runs at approximately 60 to 70 percent. In any case, the CPU should not exceed 90 percent [20, 21]. The report summarizes the CPU utilization percentage of all CPU cores used in the system during the specified period. Thus, information can be collected about CPU usage during previous specified periods, which will show us the state of the CPU and the percentage of work during those periods and within a specified amount of data rate. After we get that information, we can use the dynamic programming method to get the highest percentage of CPU usage.

Here are some readings in Table 3 about the amount of data sent in MB/S, the amount of use of the corresponding CPU, and the relationship between them. Fig. 1 illustrates the relationship between data rate and CPU utilization.

5. Proposed algorithm

Fig. 2 illustrates the proposed flowchart, and the steps of the algorithm as shown.

5.1 (first stage)

1. Start: // The algorithm begins.
2. Read n: // Read the number of data rate cases (n) from some input source. In this case study, there are 10 data rate cases.
3. Create an array of (i * i), // i = n: Create an array of size (n * n), where n is the number of data rate cases, to store intermediate results.
4. Read the data rate for each server S1, S2, ..., Sm; // m = number of servers: Read the data rate for each server (S1, S2, ..., Sm), where m is the total number of servers in the network.
5. For all i: // Start a loop over each data rate case (i).
6. For all j, // j = server number: Within the loop for each data rate case, iterate over each server (j).
7. Read the total data rate for (S1 + S2): // Read and compute the total data rate when combining servers S1 and S2. This will be useful for further calculations.
8. Read the maximum value of Matrix diameters for S1 and S2, and keep the index of the value of i and j for them: // Read the maximum values of matrix diameters for servers S1 and S2, and record their corresponding indices i and j. These indices will be helpful in the subsequent steps.
9. For each N where 1 < N < (n - (i + j) + 1): // Start another loop over each N value, where N is an index within the range 1 to (n - (i + j) + 1).
10. Read the value of data rates of S3 and keep the value index i, j, and N: // Within this N loop, read the data rate value for server S3 and record its corresponding indices i, j, and N.
11. Add the value at step 10 with the total score at step 8: // Add the data rate value obtained in step 10 with the total data rate score obtained in step 8 (from combining servers S1 and S2).
12. Maximum search value with index i, j, N: // Perform a search to find the maximum value among all the N values calculated in step 11, and record the corresponding indices i, j, and N.
13. Impose I, j, and N values to the servers S1, S2, and S3, respectively, as a new weight: // Once the maximum value and its indices are found, assign the values of i, j, and N to servers S1, S2, and S3, respectively, as their new weights. These weights are chosen to optimize the data distribution across the network.
14. The end.

5.2 Algorithm 2 (second stage)

for distributing the new requests based on the "least connection" approach with proper tie-breaking using the "round-robin" method:

1. Initialize the lists and read the inputs:

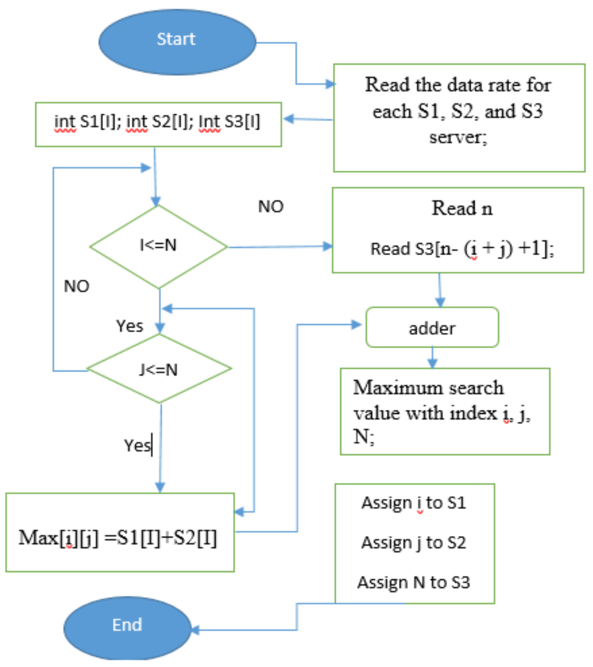

Figure. 2 Proposed flowchart (first stage)

- Number of servers (m)
- Server weights (server's weights []) - An array of size n containing the weights of each server.
- Current connections (CC []) - An array of size n containing the current connections on each server.
- Number of new requests (New requests)

2. Calculate the total connections: TC = sum of all elements in CC [].
3. Read the number of new requests to distribute (New requests).
4. Calculate the total new connections after distributing the new requests: Total new connections (TNC) = TC + New requests.
5. Calculate the total weights: TW = sum of all elements in servers weights: SW [].
6. Calculate the connections per weight: Connections per weight (CPW) = TNC / TW (rounded down to the nearest integer).
7. Initialize the list for each server: NRs [] = [_, _, ..., _] (An array of size n, initially empty).
8. Calculate the number of new requests to be assigned to each server: For each server i from 1 to m.
    - NRs[i] = round (CPW * SW [i]) – CC [i].
9. If there are remaining new requests (Total new connections are not reached): Choose multiple servers with the highest weights equal to the number of remaining requests.
    - Sort the servers in descending order based on their weights.
    - For each remaining new request, assign it to the server with the highest remaining capacity and decrease NRs [] for that server by 1.
10. Calculate the new total connections for each server: For each server i from 1 to n:
11. NC [i] = CC[i] + NRs[i].
12. Print the new total connections for each server from the NC [] list.

In situations where multiple servers can handle one more request before reaching their maximum capacity, the tie-breaking step (step 5) uses the "round-robin" method to distribute the requests among the tied servers in a sequential and balanced manner. This ensures that the load is evenly distributed among all capable servers and prevents overloading any individual server, resulting in an efficient and responsive load-balancing strategy.

5.2.1. Example usage:

Let's consider an example where we have 4 servers with their respective weights and current connections:

Number of servers (m) = 4, Server weights (SW []) = [2, 3, 5, 4], Current connections (CC []) = [1, 1, 4, 2], and the number of new requests (New requests) = 7

Now, apply the updated algorithm to distribute the new requests:

1. Total connections (TC): 1 + 1 + 4 + 2 = 8.
2. Total new requests (TNR): 7.
3. TC + TNR: 8 + 7 = 15.
4. Total weights (TW): 2 + 3 + 5 + 4 = 14.
5. Connections per weight: 15 / 14 ≈ 1.07 ≈ 1.
6. Number of new requests to be assigned to each server:
    - For Server 1: round ((1 * 2) - 1) = round (2 - 1) = 1 new request.
    - For Server 2: round ((1 * 3) - 1) = round (3 - 1) = 2 new requests.
    - For Server 3: round ((1 * 5) - 4) = round (5 - 4) = 1 new request.
    - For Server 4: round ((1 * 4) - 2) = round (4 - 2) = 2 new requests.
7. Distribute the new requests:
    - New request 1: Assign to Server 1.
    - New request 2: Assign to Server 2.
    - New request 3: Assign to Server 2.
    - New request 4: Assign to Server 3.
    - New request 5: Assign to Server 4.
    - New request 6: Assign to Server 4.
    - New request 7: Assign to Server 3. (the highest weight)

Received: September 9, 2023. Revised: November 4, 2023.

Table 4. Relations between data and CPU Utilization

Data	**CPU**	**CPU**	**CPU**
0	1	2	3
1	10	12	15
2	20	25	30
3	30	45	50
4	40	50	65
5	50	55	75
6	60	65	85
7	70	75	96
8	90	95	100
9	95	100	100
10	100	100	100

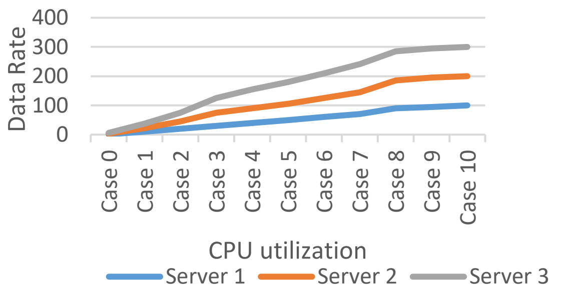

Figure. 3 Relations between data and servers

Table 5. The data rate and CPU utilization for server1

Data	0	1	2	3	4	5	6	7	8	9.	1
CPU	1	1	2	3	4	5	6	7	9	9	1
CPU	2	1	2	4	5	5	6	7	9	1	1

After distributing the seven new requests as described, the new total connections on each server will be:

- Server 1: 2 connections.
- Server 2: 4 connections.
- Server 3: 6 connections.
- Server 4: 4 connections.

Now, there are no remaining new requests to distribute, and the algorithm ensures that the distribution takes into account the server weights and the last request is correctly assigned to Server 3 as it has the highest weight.

5.3 The time complexity of the algorithm

In the given problem context, where n represents the number of data rate cases and m represents the number of servers, the time complexity is expressed as O (n^2 + m).

6. Study case

Suppose we have "n" data rate cases, where n = 10 in our study, and three different servers with different CPUs, denoted as Server1, Server2, and Server3. The corresponding data rates and CPU utilizations for these cases are presented in Table 4. It is observed that when the data rate is 0, the CPU utilization for Server1 is 1, for Server2 is 2, and for Server3 is 3. Similarly, when the data rate is 10, the CPU utilization for all three servers is 100. This pattern continues for other data rate values.

Additionally, Fig. 3 illustrates the relationships between the data rates and the servers, providing a visual representation of the data distribution among the servers.

Furthermore, Table 5 displays the specific data rate and CPU utilization values for Server 1 and Server 2 respectively. These values demonstrate the CPU utilization of Server 1 and Server 2 corresponding to different data rate cases.

In summary, the provided information highlights the data rates and CPU utilization values for various cases involving three different servers. The relationships between data rates and servers are depicted in Fig. 4, and Table 6 focuses on the data rate and CPU utilization specific to Server 1 and Server 2. Figs. 4, and 5 illustrate the relationships between the data rates with Server 1 and Server 2 respectively.

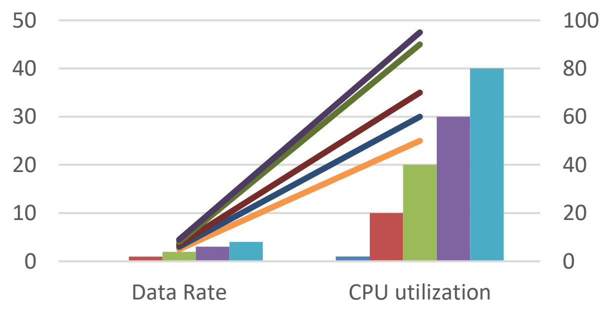

Figure. 4 Relations between data rate and CPU utilization (server 1)

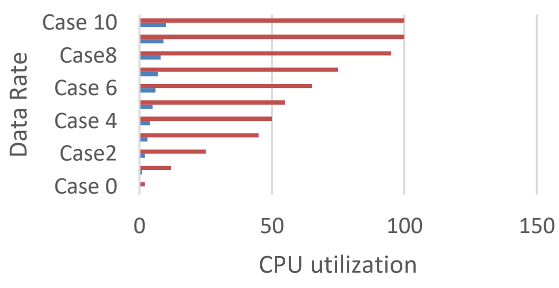

Figure. 5 Relations between data rate and CPU utilization (server 2)

Table 6. CPU utilization for Server 1 and Server 2

server1 / server2		I=0	I=1	I=2	I=3	I=4
		1	10	20	30	40
J=0	2	3*	12	22	32	42
J=1	12	13*	22	32	42	52
J=2	25	26*	35	45	55	65
J=3	45	46*	55*	65*	75*	85*
J=4	50	51	60	70	80	90
J=5	55	56	65	75	85	95
J=6	65	66	75	85	95	105
J=7	75	76	85	95	105	
J=8	95	96*	105*	115*		
J=9	100	101	110			
J=10	100	101				

I=5	I=6	I=7	I=8	I=9	I=10
50	60	70	90	95	100
52	62	72	92	97	102
62	72	82	102	107	
75	85	95	115*		
95	105*	115*			
100	110				
105					

Table 7. CPU utilization for all servers

Data rate cases	0	1	2	3	4	5	6	7	8	9	10
MAX value (X1) = (S1+S2)	3	13	26	46	55	65	75	85	96	105	115
Index of CPU utilization of server 2 and server 1 (i-j)	0-0	1-0	2-0	3-0	3-1	3-2	3-3	3-4	8-0	3-6 8-2	2-8 3-7 8-2
Server 3 Data Rate cases	10	9	8	7	6	5	4	3	2	1	0
CPU utilization server 3 (X2)	100	100	100	96	85	75	65	50	30	15	3
X1+X2	103	113	126	142	140	140	140	135	126	120	118

Table 8. Remaining CPU utilization for maximum values of servers

Data rate cases	4	5	6	7	9
MAX value (X1) = (S1+S2)	55	65	75	85	105
Index of CPU utilization of server 2 and server 1 (i-j)	3-1	3-2	3-3	3-4	3-6 8-2
Server 3 Data Rate cases	6	5	4	3	1
CPU utilization server 3 (X2)	85	75	65	50	15
X1+X2	140	140	140	135	120

The following (I*J) array (I, J= number of study data rate cases.) shows the summation of utilization for server 1 and server 2. The maximum values can be read along the diameter which is specified by a marker * from adding CPU utilization for Server 1 and Server 2.

The CPU usage values of Server 2 and Server 1 can be found in the maximum value index of Table 4.

Neglect each column with index zero for I or J or both, and also neglect each column with zero data rate, so will remove the first, second, third, fourth, ninth, and eleventh columns, the result is shown in Table 7 and the remaining CPU utilization for maximum values of servers shows in Table 8.

From the provided Table 6, there are three instances where the maximum value reaches 140. While many of these options can be considered, the most favorable choice is the second one. This preference is justified by its capability to encompass the peak state effectively. Additionally, it offers a higher level of safety compared to the initial option, where the processor operates at a high power level, potentially leading to overheating and, consequently, processor failure.

Furthermore, the second case surpasses the third case due to its potential to avoid reaching the peak state. Consequently, we collect the results from the fourth row of Table 6 for server 3 and from the index within the third row for server 2 and server 1.

Received: September 9, 2023. Revised: November 4, 2023.

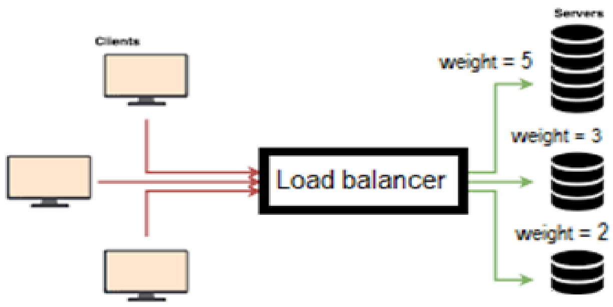

Figure. 6 Proposed weight of servers

- Option 1:
  - 6 to Server 3, 3 to Server 2, and 1 to Server 1 or
- Option 2:
  - 5 to Server3, 3 to Server2, and 2 to Server1 or
- Option 3:
  - 4 to Server3, 3 to Server2, and 3 to Server1.

The second option is therefore the best that gives maximum throughput without any risks to the state of the CPU as a result of the voltage that may cause high temperatures in the working State at or near the maximum power. So, the weight of servers 1, 2, and 3 will be 5, 3, and 2 respectively. Fig. 6 illustrates the weight of the servers.

The number of connections will be taken into account, for example, now suppose we have ten requests. This calculation of orders is divided by the total weight, 10 /2+3+5 = 10 /10 =1 this means one request per one weight, so that, two requests to server 1, three requests to server 2, and five requests to server 3, and so on. Also, the algorithm of the second stage will take care of the old connection as illustrated in the above example 6.1.

7. Result and discussed

While having the same data rate request job and working with 100% CPU utilization might indicate similar workload characteristics, it does not necessarily mean that the servers have the same properties. Several other factors can affect server performance and properties:

1. Hardware specifications: The servers might have different hardware specifications, such as CPU type, number of cores, memory size, storage type, and network interfaces. These variations can lead to different levels of performance even when working at full CPU utilization.
2. Network latency: The servers could be located in different geographical locations, and network latency between them and the clients can vary. This can affect response times and overall user experience.
3. Software configuration: Differences in software configurations, operating systems, and software versions can impact server performance. Additionally, variations in how the software is optimized and configured can lead to different results.
4. Load balancing and traffic distribution [22, 23]: Even if all three servers are handling the same data rate request job, the load balancing and traffic distribution algorithms in place might differ, which can affect how requests are distributed among the servers.
5. Power and cooling: The servers might be housed in different environments with varying power and cooling capabilities. This can influence their reliability, stability, and long-term performance.
6. Scalability: The ability of each server to scale and handle increased loads in the future might differ. Some servers might have better scalability options than others.
7. Redundancy and fault tolerance: The level of redundancy and fault tolerance built into the server infrastructure can be different, impacting the overall reliability and availability.
8. Security features: Servers may have different security measures implemented, which can affect their vulnerability to attacks and their ability to protect data [24].

While having a similar data rate request job and 100% CPU utilization might indicate comparable performance under the current conditions, it's essential to consider these other factors to understand the overall properties and capabilities of the servers. Proper benchmarking and performance testing can help in assessing the differences and similarities between the servers more accurately.

To determine which server might have a better classification for the given data rate and CPU utilization, we need to analyze the data and understand the relationship between the two variables. One common approach to analyzing such data is to plot it on a graph and observe the trends. As in Fig. 4. The resulting plot will show three lines representing the CPU utilization for each server at different data rates. By observing the plot, we can make some general observations:

1. Server 1: It shows a relatively linear increase in CPU utilization with data rate, but the slope of the line is less steep compared to the other servers.
2. Server 2: It also shows a linear increase in CPU utilization with data rate, and the slope of the line is steeper than that of Server 1.
3. Server 3: It demonstrates a nearly constant CPU

utilization, which remains close to 100% regardless of the data rate.

Based on the plot, it seems that Server 3 has the highest CPU utilization consistently, indicating that it might be processing the data rate requests more efficiently than the other servers. However, the classification of which server is "better" might depend on specific criteria or requirements for the system. For example, if the goal is to maximize CPU utilization, then Server 3 would be considered the best. On the other hand, if the goal is to have more linear scalability with data rate, Server 1 or Server 2 might be preferred. this analysis is based solely on the data provided, and other factors such as the specific workload, server hardware, and software configurations should also be taken into account to make a more informed decision about server classification.

The fact that Server 3 reached 100% CPU utilization earlier than the other servers could be due to several reasons. While Server 3 might have achieved higher CPU utilization faster, it doesn't necessarily mean it is better in all scenarios. Some potential reasons for this behavior:

1. Processing efficiency: Server 3 could have a more efficient processing mechanism, allowing it to handle data rate requests more quickly and consume CPU resources at a faster rate. This efficiency might be advantageous in certain situations where high-speed processing is required.
2. Limited resources: Server 3 might have fewer CPU cores or overall computing resources compared to the other servers. As a result, it reaches 100% CPU utilization sooner because it has fewer resources available to handle the increasing data rate requests.
3. Optimization differences: Each server may have different software configurations and optimizations. Server 3 might be configured to prioritize speed over scalability, leading to faster CPU utilization saturation.
4. Load balancing: The system might be using a load balancing algorithm that directs a higher proportion of data rate requests to Server 3. This can lead to faster CPU utilization saturation on that particular server.
5. Workload characteristics: The data rate requests might have certain patterns or characteristics that make Server 3's processing capabilities particularly suitable for handling them efficiently.
6. Bottlenecks: Server 3 could be facing other bottlenecks, such as memory limitations or disk I/O constraints, which cause the CPU to reach maximum utilization earlier even though there might still be processing capacity left.

Higher CPU utilization does not always equate to better performance or efficiency. A server operating at 100% CPU utilization is typically running at full capacity, leaving little room for handling additional spikes in workload or processing unexpected events. In some cases, having some headroom in CPU utilization can improve system responsiveness and robustness.

Determining which server is "better" depends on the specific use case, workload requirements, and overall system design. While Server 3 might excel in certain situations, Servers 1 and 2 could be more suitable for other scenarios where linear scalability and flexibility are essential. A comprehensive analysis of the servers' performance under different workloads and scenarios is necessary to make a more informed decision about which server is truly better for a particular use case.

In this proposed paper, the data rate is a basis for assigning weights to the servers in a weighted round-robin algorithm. In a weighted round-robin approach, servers are assigned different weights based on their capabilities and performance characteristics. The higher the weight assigned to a server, the more frequently it will receive requests in comparison to servers with lower weights.

Using the data rate as a factor to determine the weights can be a reasonable strategy, especially if you want to take into account the current load and processing capacity of each server. The data rate can be a proxy for the current workload or demand on each server. Servers that can handle higher data rates might be assigned higher weights to receive more requests, while servers with lower data rates might have lower weights and receive fewer requests.

The general outline of how can incorporate the data rate into the weighted round-robin.

1. Calculate weights: Calculate the weights for each server based on their data rate. One simple way to do this is to assign weights proportionally to the data rate. For example, if Server A has a data rate of 50 and Server B has a data rate of 100, you could assign a weight of 1 to Server A and a weight of 2 to Server B. but the drawback of this way is not considered the ability of the servers of other parameters and data rate may change over time.
2. Dynamic updates: Since the data rate may change over time due to varying workloads, you might want to consider dynamically updating the

weights at regular intervals. This way, the server selection adapts to the changing data rates and load distribution, as in our proposed way.

Using the data rate alone as the sole criterion for assigning weights might not cover all relevant server performance aspects. Other factors, such as server capacity, CPU utilization, memory availability, and response times, should also be taken into account for a more comprehensive load-balancing strategy. therefore, employing the least connection algorithm with the data rate is a perfect criterion for assigning weights and will cover most of the relevant server performance aspects.

Overall, utilizing the data rate to assign weights in a weighted round-robin algorithm can be a useful approach to achieve load balancing based on the current workload and capacity of the servers. However, it's essential to monitor and fine-tune the weights based on real-world performance and usage patterns to optimize the load-balancing strategy for specific use cases.

8. Conclusion

Numerous researchers have dedicated their efforts to designing specialized weight metrics for groups of servers to ensure a fair distribution of workloads, aligning these weights with the capacity of each server. Different studies have explored diverse metrics, including memory size, CPU processing speed, and the number of links associated with the servers. Each metric has aimed to address the load-balancing challenge effectively. The availability of sufficient bandwidth plays a pivotal role in determining the efficiency of data transfer within the network. Additionally, factors such as server processing speed, response time, disk usage, and latency speed within the server itself significantly impact overall performance. These crucial determinants ultimately influence the data transfer rate. In this research, we adopt the data transfer rate as the primary metric to calculate the server weights. By doing so, we aim to create weights that accurately reflect the actual operational environment of the servers. This approach ensures a well-balanced and realistic representation of the workload distribution, optimizing the overall performance of the data network.

To enhance the least connection algorithm for distributing requests among several servers, we prioritize selecting the server with the fewest active connections while also ensuring that the current connections on the servers are taken into account. By doing so, we can achieve a more balanced load distribution, avoiding any neglect of the existing connections on the servers.

Conflicts of interest

There is no conflict of interest regarding the publication of this paper.

Author contributions

1. Mahdi S. Almhanna proposed the methodology, authored the main manuscript text, and prepared all figures except (the flowchart in Fig. 3), as well as all the tables.
2. Tariq A. Murshedi provided valuable insights into the proposed methodology, prepared Fig. 3, and reviewed the manuscript.
3. Ahmed M. Al-Salih contributed the idea for Algorithm 2.
4. Rafah M. Almuttairi conducted all the experiments. Also, checked and improved the English language throughout the manuscript, and also supervised the various steps of its development.

Acknowledgements

The authors express their gratitude to the editors and reviewers for their valuable and constructive feedback. Additionally, we would like to extend our thanks and appreciation to the University of Babylon and the College of Information Technology for their unwavering support of the staff.

References

[1] A. A. Neghabi, N. J. Navimipour, M. Hosseinzadeh, and A. Rezaee, "Load Balancing Mechanisms in the Software Defined Networks: A Systematic and Comprehensive Review of the Literature", *IEEE Access*, Vol. 6, pp. 14159-14178, 05 March 2018.

[2] D. K. Patel, D. Tripathy, and C. R. Tripathy, “Survey of load balancing techniques for Grid”, *Journal of Network and Computer Applications*, Vol. 65, pp. 103-119, 24 February 2016.

[3] W. Wang and G. Casale, "Evaluating Weighted Round Robin Load Balancing for Cloud Web Services", In: *Proc. of 2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, Timisoara, Romania, pp. 393-400, 2014.

[4] A. Gajbhiye and D. S. Singh, “Global Server Load Balancing with Networked Load Balancers for Geographically Distributed Cloud Data-Centers”, *International Journal of*

Computer Science and Network, Vol. 6, No. 6, pp. 682-688, 2017.

[5] R. M. Almuttairi, R. Wankar, A. Negi, R. R. Chillarige, and M. S. Almahna, "New replica selection technique for binding replica sites in Data Grids", In: *Proc. of 2010 1st International Conference on Energy, Power, and Control*, Basrah, Iraq, pp. 187-194, 2010.

[6] X. Zhang, Y. Qu, and L. Xiao, "Improving distributed workload performance by sharing both CPU and memory resources", In: *Proc. of 20th IEEE International Conference on Distributed Computing Systems*, Taipei, Taiwan, pp. 233-241, 06 August 2002.

[7] A. Jangra and N. Mangla, "An efficient load balancing framework for deploying resource scheduling in cloud based communication in healthcare", *Measurement: Sensors*, Vol. 25, p. 100584, 2023.

[8] S. Afzal and G. Kavitha, “Load balancing in cloud computing – A hierarchical taxonomical classification”, *Journal of Cloud Computing*, Vol. 8, No. 1, 2019.

[9] M. S. Almhanna, F. S. A. Turaihi, and T. A. Murshedi, “Reducing waiting and idle time for a group of jobs in the grid Computing”, *Bulletin of Electrical Engineering and Informatics*, Vol. 12, No. 5, pp. 3115-3123, 2023,

[10] M. S. Almhanna, "Minimizing server idle time", *Annual Conference on New Trends in Information & Communications Technology Applications*, Baghdad, Iraq, pp. 128-131, 2017.

[11] A. A. Adewojo and J. M. Bass, “A novel weight-assignment load balancing algorithm for cloud applications”, *SN Computer Science*, Vol. 4, 2023.

[12] S. L. Chen and Y. Y. Chen, and S. H. Kuo, “Clb: A novel load balancing architecture and algorithm for cloud services”, *Computers and Electrical Engineering*, Vol. 58, No. 4, PP. 154-60, February 2017.

[13] A. Mohammed, N. F. Abdullah, S. Alani, O. S. Alheety, M. M. Shaker, M. A. Saad, and S. N. Mahmood, "Weighted Round Robin Scheduling Algorithms in Mobile AD HOC Network", In: *Proc. of 2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, Ankara, Turkey, pp. 1-5, 2021.

[14] D. A. Shafiq, N. Z. Jhanjhi, A. Abdullah, and M. A. Alzain, "A Load Balancing Algorithm for the Data Centres to Optimize Cloud Computing Applications", *IEEE Access*, Vol. 9, pp. 41731-41744, 2021.

[15] P. Somwang, “Efficient Load Balancing for Cloud Computing by Using Content Analysis", *International Journal of Communication Networks and Information Security*, Vol. 12, No. 2, August 2020.

[16] E. H. M. M. Cruz, M. Diener, L. L. Pilla, and P. O. A. Navaux, "EagerMap: A Task Mapping Algorithm to Improve Communication and Load Balancing in Clusters of Multicore Systems", *ACM Transactions on Parallel Computing*, Vol. 5, No. 4, pp 1–24, 2019.

[17] H. Son, S. Lee, S. Kim, and Y. Shin, "Soft Load Balancing Over Heterogeneous Wireless Networks", *IEEE Transactions on Vehicular Technology*, Vol. 57, No. 4, pp. 2632-2638, 2008.

[18] H. Babbar, S. Parthiban, G. Radhakrishnan, and S. Rani, "A genetic load balancing algorithm to improve the QoS metrics for software-defined networking for multimedia applications", *Multimedia Tools and Applications*, Vol. 81, No. 7, pp. 9111-9129, 2022.

[19] A. Tarek, H. Elsayed, M. Rashad, M. Hassan, and P. E. Kafrawy, "Dynamic Programming Applications: A Survey", In: *Proc. of 2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, Giza, Egypt, pp. 380-385, 2020.

[20] S. A. Abbas and M. S. Almhanna, “Distributed Denial of Service Attacks Detection System by Machine Learning Based on Dimensionality Reduction”, *2021 J. Phys.: Conf. Ser*, Babylon-Hilla City, Iraq, Vol. 1804, pp. 1-13, March 2021.

[21] J. Zhou, Y. Zhang, L. Sun, S. Zhuang, C. Tang and J. Sun, "Stochastic Virtual Machine Placement for Cloud Data Centers Under Resource Requirement Variations", *IEEE Access*, Vol. 7, pp. 174412-174424, 2019.

[22] J. C. Patni and M. S. Aswal, "Distributed load balancing model for a grid computing environment", In: *Proc. of 2015 1st International Conference on Next Generation Computing Technologies*, Dehradun, India, pp. 123-126, 2015.

[23] M. H. Balter and A. B. Downey, A.: “Exploiting process lifetime distributions for dynamic load balancing”, *ACM Transactions on Computer Systems*, Vol. 15, No. 3, pp. 253–285, 1997.

[24] M. Balanici and S. Pachnicke, "Classification and forecasting of real-time server traffic flows employing long short-term memory for hybrid E/O data center networks", *IEEE/OSA Journal of Optical Communications and Networking*, Vol. 13, pp. 85-93, February 2021.