INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY

RESEARCH ARTICLE

An Optional Memory Balancing Algorithm for Big Data Based on Distributed File System

Mukunzi Thomas[1]*, **Huang Dongjun**[1], **Nzaramba Antoine**[1]

1 Central South University, School of Computer Science and Technology, Department of Computer Science, China

OPEN ACCESS

Received: 17-03-2023

Accepted: 29-07-2023

Published: 30-09-2023

Citation: Thomas M, Dongjun H, Antoine N (2023) An Optional Memory Balancing Algorithm for Big Data Based on Distributed File System. Indian Journal of Science and Technology 16(37): 3043-3049. https://doi.org/10.17485/IJST/v16i37.611

* **Corresponding author**.

mukmaforo@gmail.com

Funding: None

Competing Interests: None

ISSN
Print: 0974-6846
Electronic: 0974-5645

Abstract

Background/Objectives: Load balancing algorithms are a type of algorithm that assists with efficient workloads spread. To reduce server overload, increase resource usage, lower latency, and maximize throughput, efficient distribution is required. There are several load balancing algorithms, all of these algorithms have flaws that render them ineffectual in many real-world settings. The current paper aims to improvise the algorithm by proposing a heuristic to decide the next job to be allotted to reduce the net response time. This is achieved by storing the incoming jobs in a priority queue (heap) and using ageing to prevent starvation. **Methods:** In the current research, we assign the incoming client request to the virtual machine such that estimated finish time of the service is minimum. We calculate finishTime for all the virtual machines, then the VM which gives the minimum finishTime is assigned the incoming client request. Once a client request is assigned to a VM, it gets added to the virtual Machine Queue. The virtual Machine Queue behaves like a priority queue; it gives priority to the task which has the least instruction Count. This ensures that the Net Response Time of the load balancer is minimized. **Findings:** Significant improvements have been achieved with respect to response time and throughput. We calculated the Response Time and the Throughput Time for the Load Balancing Using Estimated Finish Time Algorithm compared to IEFTA. The numbers of client requests were varied by a factor of 10, starting from 10, and we got an efficient distribution. **Novelty:** By using our proposed Algorithm Pseudocode of Load Balancing Algorithm, different coefficient of VMs were calculated. We compared the Estimated Finish Time Algorithm to IEFTA, we suggest an approach that addresses this limitation while also improving responsiveness and processing time.

Keywords: Cloud Computing; Distributed Systems; Load Balancing; Virtual Machines; Data Migration

1 Introduction

Load balancing is a strategy for offloading task distribution across linked nodes in a parallel and/or distributed computing environment in order to optimize system performance. Load balancing is required to guarantee efficient resource usage and the absence of overloaded servers. Load balancing may be divided into two categories: static and dynamic. While static load balancing does not consider the condition of the system (such as node performance) while making judgments regarding job allocation, dynamic load balancing does. Static load balancing algorithms operate based on previous knowledge of the services and their overall condition, resource requirements, and communication time(1). They are preferable in cases when the work queue's behavior is predictable and jobs are comparable in nature. They have a shorter processing time and are less sophisticated than dynamic load balancing methods(2). Dynamic load balancing algorithms(3), on the other hand, respond spontaneously to changes in the system. They often employ a master-slave architecture in which the master delegate's incoming processes to slaves based on the condition of the environment. Load balancing is performed at two levels-host level (known as Virtual Machine (VM) scheduling policy) and VM level (known as task scheduling policy)(4). For more details on load balancing, we advise readers to the huge number of papers(2,5–7). Furthermore, decisions can be made at either level to pick between Time-Shared (TS) scheduling and Space-Shared (SS) scheduling.

There is a dynamic load balancing algorithm based on estimating the finish times of the jobs. It builds upon Active Monitoring Load Balancer (AMLB) algorithm(8). The readers can refer also to(9–13) for a detailed information on load balancing algorithm and load balancing for effectively distributed in-memory based on computing. AMLB maintains a datacenter broker which logs information and current activity status of each VM(14). Upon the arrival of a job, it identifies the least loaded VM and assigns the job to it. Although the algorithm improves the response time and works fine to reduce bottleneck scenarios, it does not consider the job size and actual instant powers of VMs(2). To counter this issue, an algorithm was proposed to take both the missing factors into consideration. It demarcates four scheduling scenarios as follows and aims to improve the processing time and response time:

a) SS in host level and SS in VM level;
b) SS in host level and TS in VM level;
c) TS in host level and SS in VM level;
d) TS in host level and TS in VM level.

The algorithm utilizes a datacenter broker to maintain the status of all VMs. While there is a job available in the queue to be allocated, it calculates the finish time of the job for each VM using the formulae discussed later. Finally, it picks the VM with the earliest completion time and assigns the task to it. The proposed algorithm has a shortcoming that although it considers the earliest finish time which can be offered to a client request by a virtual machine, it fails to schedule the request within the virtual machine queue effectively. It stores the incoming jobs in a queue and processes them in a First in First Out (FIFO) fashion. This does not allow much room to decide upon as to which job should be processed next. This paper aims to improvise the algorithm by proposing a heuristic to decide the next job to be allotted to reduce the net response time. This is achieved by storing the incoming jobs in a priority queue (heap) and using ageing to prevent starvation.

The following section analyses some of the load balancing algorithms:

1.1 Round Robin

Round robin load balancing is effective when all the servers have similar compute capabilities, specifications and storage capacity. In this, the load balancer assigns the client requests to the servers in a specific sequence. Upon reaching the end of the sequence, it loops back to be beginning, i.e., client requests are handled in a cyclic manner.

The main advantage of this algorithm is that it is effective if servers are homogeneous in nature, and it is extremely easy to implement such a load balancer. However, it is common to have heterogeneous systems in place, in such cases the following variants are useful:

- Weighted Round Robin: Each of the servers is assigned a weight depending on the load-handling capacity of the server, decided using an adaptive algorithm or by the system administrator. A server with higher weight is assigned a larger portion of the client requests.
- Dynamic Round Robin: The servers are assigned weights dynamically depending on the server's current capacity and idleness. Then, the client requests are assigned to a server, proportional to its weight.

1.2 Least connection

A least connection load balancer assigns a client request to the server which has the least number of active connections. This is a dynamic load balancing algorithm since it makes use of the server's current state, rather than following an arbitrary sequence, unlike round robin approaches. The main disadvantage of such an approach is that the current server load is not taken into consideration while distributing requests, this leads to degradation of performance of the load balancer. There exists a weighted variant for least connection load balancing algorithm as well. In weighted least connection load balancing, a numerical weight is assigned to each server depending on the compute capability of the server. If multiple servers have the same number of active connections, then the server with higher weight is chosen to process the request.

1.3 Mapping Policy based Load balancing

The authors[15] proposed a load balancing algorithm based on virtual machine to physical machine mapping. The algorithm contains a resource monitor and a scheduling con-troller. The resource monitor tracks the usage, idleness, and availability of the resources. Whereas the scheduling controller does the task of assigning requests to the appropriate server.

1.4 Load balancing using Ant colony optimization

In[10], the authors suggested a probabilistic load balancing algorithm which makes use of ant-colony optimization (ACO). In the approach proposed by the paper, the ants keep track of overloaded and under loaded nodes in the network, and the pheromone table is updated. The pheromone table gives an estimate of resource utilization by each node.

The traversal is done using two types of movements:

- **Forward movement:** The ants continuously move in the forward direction in the network and keep track of over-loaded and under loaded nodes.
- **Backward movement:** If an ant encounters an overloaded node and it has previously encountered a under loaded node, it redistributed the work. The process can be vice-versa as well.

2 Methodology

2.1 Proposed algorithm

In the proposed algorithm, we assign the incoming client request to the virtual machine such that estimated finish time of the service is minimum. To do this, we calculate finishTime according to formula (2) and (3) for all the virtual machines. The VM which gives the minimum finishTime is assigned the incoming client request. Once a client request is assigned to a VM, it gets added to the virtual Machine Queue. The virtual Machine Queue behaves like a priority queue; it gives priority to the task which has the least instruction Count. This ensures that the Net Response Time (9) of the load balancer is minimized.

Response time for a job with arrival time arrivalTime, finish time finishTime and transmission delay TDelay is given as:

$$Response\ Time\ =\ finish\ Time\ -\ arrival\ Time\ +\ T\ Delay \tag{1}$$

Assuming startTime (x) as the time when the job x started, time as the current simulation time, instructionCount as the total number of instructions of a job, core (x) as the number of cores or processing elements required by the job, capacity as the average processing capacity (in MIPS) of a core for a job, we obtain following equations:

If the host scheduling policy is SS-SS or TS-SS, finishTime is calculated as:

$$finish\ Time = start\ Time\ (x) + \frac{instruction\ Count}{capacity\ \times core\,(x)} \tag{2}$$

If the host scheduling policy is SS-TS or TS-TS, finishTime is calculated as:

$$finish\ Time = time + \frac{instruction\ Count}{capacity\ \times core(x)} \tag{3}$$

Execution time of a job is calculated as:

$$Execution\ Time = \frac{instruction\ Count}{capacity\ \times core(x)} \tag{4}$$

Capacity in each case is calculated as follows:

1) VM scheduling policy is SS, task scheduling policy is SS

$$Capacity = \sum_{i=1}^{np} \frac{Cap(i)}{np} \tag{5}$$

where, $Cap(i)$ is the processing power of the core$i$, $np$is the number of real cores of the considered host.

2) VM scheduling policy is SS, task scheduling policy is TS:

$$Capacity = \frac{\sum_{i=1}^{np} Cap(i)}{max(\sum_{j=1}^{\alpha} cores\,(j)\,, np))} \tag{6}$$

where, $cores(j)$ are number of cores that job j needs. α is total job in VM which contains the job.

3) VM scheduling policy is TS, task scheduling policy is SS:

$$Capacity = \frac{\sum_{i=1}^{np} Cap(i)}{max(\sum_{k=1}^{\beta} \sum_{j=1}^{\gamma} cores\,(j)\,, np))} \tag{7}$$

where, β is the number of VMs in the current host, γ is number of jobs running simultaneously in VM_k.

4) VM scheduling policy is TS, task scheduling policy is TS:

$$Capacity = \frac{\sum_{i=1}^{np} Cap(i)}{max(\sum_{j=1}^{\delta} cores\,(j)\,, np))} \tag{8}$$

where, δ is total job of the considered host.

However, this approach creates a problem, if a task has extremely low priority it is likely that it will never get scheduled. This problem is commonly known as starvation. The proposed algorithm resolves this issue by using ageing. In ageing a client request's priority is artificially increased by an amount quantified by its wait time. This ensures that upon passage of time, even a request with low instructionCount will eventually gain sufficient priority to get scheduled. Such an approach guarantees low response time and no starvation.

The Net Response Time for the load balancer is finally calculated using Response Time of each request as follows:

$$\text{NetResponseTime} = \sum_{i=1}^{\text{requestCount}} \frac{\text{ResponseTime }(i)}{\text{requestCount}} \tag{9}$$

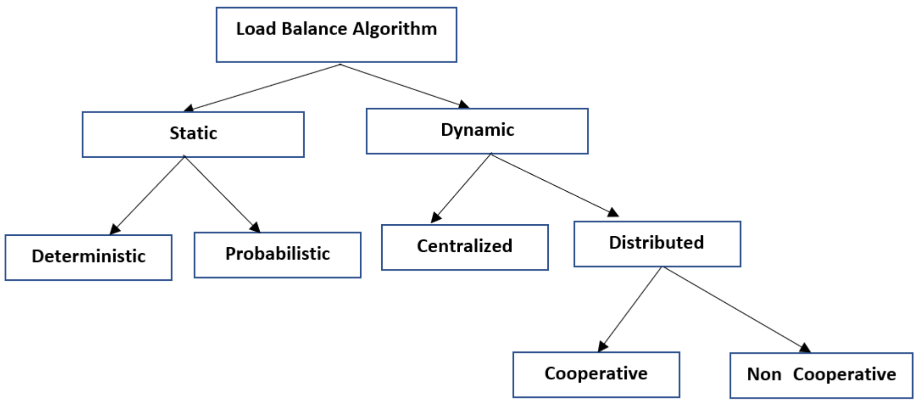

Fig 1. Load Balancing Algorithm Diagram

3 Results and Discussion

The results are shown and described according to the figures and tables, but there is no discussion that contrasts the results with related works, so it is suggested that the authors should include a discussion of the results duly compared contrasted with related studies.

3.1 Experimental results

Significant improvements have been achieved with respect to response time and throughput. Metrics have been logged in the tables below. To simulation, a variable number of cores were used (ranging upto 5) per VM. The numbers of client requests were varied by a factor of 10, starting from 10, as indicated in the tables. The algorithm was implemented using C++ (gcc version 7.5.0). The experiments were run on a system having Intel® Core^{TM} i7-8550U CPU @ 1.80GHz × 8 processor.

Table 1. Response Time: Load Balancing Using Estimated Finish Time Algorithm versus IEFTA

Number of Client Requests	Estimated Finish Time Algorithm	IEFTA
10	14.944537	14.944537
10^2	38.853931	36.879389
10^3	434.767279	313.244709
10^4	4549.779616	3205.598167
10^5	45868.511618	32449.084248

Table 2. Throughput Time: Load Balancing Using Estimated Finish Time Algorithm versus IEFTA

Number of Client Requests	Estimated Finish Time Algorithm	IEFTA
10	0.05614448	0.05614448
10^2	0.02767749	0.03199268723
10^3	0.00228050171	0.00396647467
10^4	0.00021972473	0.0003918567
10^5	0.00002177804	0.0000384324

Table 3. Pseudocode of Load Balancing Algorithm

Pseudocode of load balancing algorithm

```
1. Input: The set task; the set VM
2. While there are tasks in the list do
3. For all VMs of a host do
4. The $C_i$ and $L_{VMj,I}$ of every VMs are calculated
5. The $PT_i$ and PT of every VMs are calculated
6. The $\delta$ of every VMs is calculated
7. If $\delta \leq$ T, then
8. System is balanced, and Send Task to Partition;
9. The $Pos_j$ of every VMs is calculated
10. The $P_j$ and $CT_i$ of every tasks are calculated
11. If $Pos_{jmax}$ and $CT_{imax}$
12. Assign $task_i$ to VM
13. Exit
14. If L > maximum capacity
15. Load balancing is not possible
16. Else
17. Trigger loading balancing
18. End For
19. End while
```

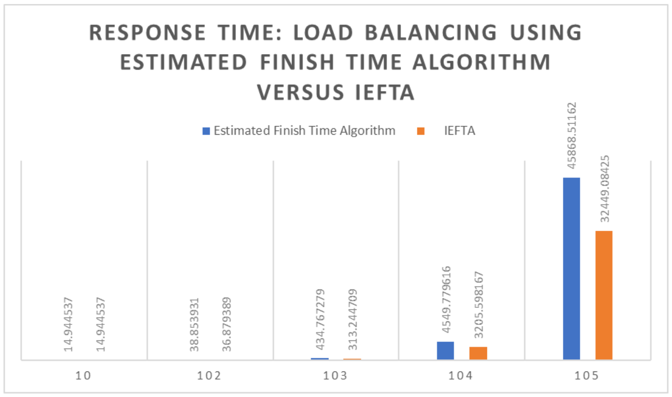

Fig 2. Response Time: Load Balancing using Estimated Finish Time Algorithm versus IEFTA

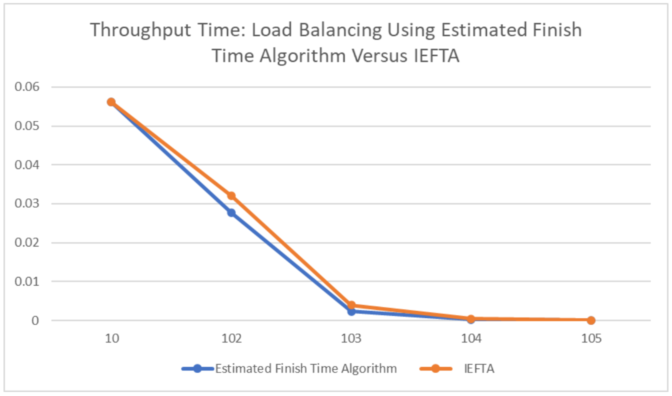

Fig 3. Throughput: Load Balancing using Estimated Finish Time Algorithm versus IEFTA

4 Conclusion

Conclusion must be pepped up with novelty connecting with objectives of the study. It should also be embedded with quantitative data to buttress the claim. Conclusion should not look verbose. Load balancing aids in the efficient use of cloud computing resources and has an impact on the issue of data center power consumption. Within virtual machines, the study provides an effective load balancing algorithm based on the concept of forecasting the end of service time paired with a shortest task first queue. To avoid famine in diverse cloud computing settings, it is complemented with ageing. We investigated scheduling situations of space sharing (SS) at the host level and space sharing (SS) at the VM level, and a set of equations were constructed to compute the virtual core's average reaction time and throughput. The simulation results demonstrated that the suggested method is more effective, with shorter processing and reaction times when compared to existing state-of-the-art techniques. Such an effective approach will result in more efficient use of computer resources.

References

1) Joshi V. Load Balancing Algorithms in Cloud Computing. *International Journal of Research in Engineering and Innovation*. 2019;3:530–532. Available from: https://hal.archives-ouvertes.fr/hal-02884073.
2) Bok K, Choi K, Choi D, Lim J, Yoo J. Load Balancing Scheme for Effectively Supporting Distributed In-Memory Based Computing. *Electronics*. 2019;8(5):546. Available from: https://doi.org/10.3390/electronics8050546.
3) Patel KD, Bhalodia TM. An Efficient Dynamic Load Balancing Algorithm for Virtual Machine in Cloud Computing. In: 2019 International Conference on Intelligent Computing and Control Systems (ICCS). IEEE. 2019;p. 145–150. Available from: https://doi.org/10.1109/ICCS45141.2019.9065292.
4) Xia H, Liu M, Chen Y, Jin X, Wang Z, Wang F. A Load Balancing Strategy Of "Container Virtual Machine" Cloud Microservice Based On Deadline Limit. *2022 14th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*. 2022;p. 998–1002. Available from: https://doi.org/10.1109/ICMTMA54903.2022.00202.
5) Chen S, Zhou M. Evolving Container to Unikernel for Edge Computing and Applications in Process Industry. *Processes*. 2021;9(2):351. Available from: https://doi.org/10.3390/pr9020351.

6) Murugan S, Jeyalaksshmi S, Mahalakshmi B, Suseendran G, Jabeen TN, Manikandan R. Comparison of ACO and PSO algorithm using energy consumption and load balancing in emerging MANET and VANET infrastructure. *Journal of critical reviews*. 2020;7(09). Available from: https://doi.org/10.31838/jcr.07.09.219.
7) Priya N, Priya SS. An Experimental Evaluation of Load Balancing Policies Using Cloud Analyst. In: Lecture Notes in Networks and Systems. Springer Nature Singapore. 2022;p. 185–198. Available from: https://doi.org/10.1007/978-981-16-7657-4_16.
8) Ijeoma CC, Inyiama P, Samuel A, Okechukwu OM. Review of Hybrid Load Balancing Algorithms in Cloud Computing Environment. . Available from: https://arxiv.org/abs/2202.13181.
9) Ibrahim IA, Bassiouni M. Improvement of job completion time in data-intensive cloud computing applications. *Journal of Cloud Computing*. 2020;9(1). Available from: https://doi.org/10.1186/s13677-019-0139-6.
10) Pradhan P, Ku P, Ray BA. A New Load Balancing Algorithm for Virtual Machine Allocation in Cloud Computing. 2020.
11) Suseendran G, Chandrasekaran E, Akila D, Kumar AS. Banking and FinTech (Financial Technology) Embraced with IoT Device. *Data Management, Analytics and Innovation*. 2020;1:197–211. Available from: https://doi.org/10.1007/978-981-32-9949-8_15.
12) Ekwonwune EN, Ezeoha BU. Scalable Distributed File Sharing System: A Robust Strategy for a Reliable Networked Environment in Tertiary Institutions. *International Journal of Communications, Network and System Sciences*. 2019;12(04):49–58.
13) Xing QJJ, Shen YYY, Cao R, Zong SX, Zhao SX, Shen YFY. Functional movement screen dataset collected with two Azure Kinect depth sensors. *Scientific Data*. 2022;9(1). Available from: https://doi.org/10.1038/s41597-022-01188-7.
14) Jyoti A, Shrimali M. Dynamic provisioning of resources based on load balancing and service broker policy in cloud computing. *Cluster Computing*. 2020;23(1):377–395. Available from: https://doi.org/10.1007/s10586-019-02928-y.
15) Sanjaya KP, Prasanta KJ. Load balanced task scheduling for cloud computing: a probabilistic approach. 2019. Available from: https://link.springer.com/article/10.1007/s10115-019-01327-4.