Load Balancing in Cloud Computing Using Multi-agent-Based Algorithms

Shyama Barna Bhattacharjee

Abstract In distributed environments, cloud computing is widely used to manage user requests for resources and services. Resource scheduling is used to handle user requests for resources based on priorities within a given time frame. The resource scheduling concept of a virtual machine is used to manage servers. To manage load and perform resource scheduling in a smooth manner in this paper, an attempt has been made to propose a multi-agent-based load-balancing mechanism. In a multi-agent-based load-balancing mechanism, multiple agent nodes at different layers are used to manage load and requests and increase throughput.

Keywords Cloud computing · Agent · Multi Agent · Virtual machine and load balancing

1 Introduction

Cloud computing is one such computing model that is so popular among this generation for providing better services to its users. It highly depends upon virtualization technology [1] as well as the internet. In the cloud of Infrastructure-as-a-Service (IaaS), an application from the artificial resource pool is placed in a virtual machine (VM). VMs having common machines have the tendency to run several operating environments of an operating system at one time. It can also be configured to provide several services and software in accordance with several service requests. Additionally, VMs can be initiated and ceased on one machine. It can also be transmitted between different machines, which hugely expand the elasticity of the resource allocation. The issues of over provisioning [2] or under provisioning appear because of the unpredictability of the resource usage of applications by users. For maximising the infrastructure utilisation without breaching the Service Level Agreement (SLA)

S. B. Bhattacharjee (✉)
Department of CSE, UIET, Kurukshetra University, Kurukshetra, Haryana, India
e-mail: shyamabarna02@gmail.com

S. Jain et al. (eds.), *Emergent Converging Technologies and Biomedical Systems*,
Lecture Notes in Electrical Engineering 1040,
https://doi.org/10.1007/978-981-99-2271-0_23

[3], the data centre workload can be easily balanced through the VM migration process.

State algorithms are such existing migration methods that choose suitable node for migration depending upon the recent execution and information of hardware workload [4]. One such traditional static VM migration method is the dual-threshold method [5] that keeps both a lower bound and an upper bound on host machines' workload and initiates the migration when the workload is below a lower bound or over an upper bound. Unfortunately, these strategies of threshold-based migration could not foresee the host machines' upcoming workload trend. Additionally, if the host machine's workload increases even for a second (for any reason), irrelevant and useless VM migration would be triggered. Different and useful methods finalise decisions depending upon the recent status as well as the past information of workload [6].

2 Load Balancing in Cloud Computing

To enhance the performance as well as the utilisation of resources in the system, dividing the employment among the accessible processors is all that is required. The aim of readjusting load is to dispense the load among various nodes in different cloud settings. Load adjustment is essentially an attempt to carefully balance the overall system of load in multiple nodes by transmitting employment between nodes [7]. This is done in order to carefully dispense the ready nodes to give satisfactory overall performance of the system [8].

2.1 Load-Balancing Taxonomy

(i) *Static balancing of load*: It is an algorithm basically to ensure the system's typical behaviour. Mostly, static algorithms are not complex and are ineffective because they fail to reflect on the current and most recent state of the system [9].

(ii) *Dynamic balancing of load*: Dynamic algorithms dispense load between the nodes by utilising the most recent node state. Since dynamic load balancing has to gather as well as respond to the state of the system, it is more complex than static load balancing. Additionally, they have the tendency to behave in an extra efficient manner as compared to the state algorithm because of the dynamic load-balancing considerations [10].

2.2 Agent Based Load Balancing

(i) *Agent*: An agent is basically a process entity which usually acts for other objects by performing tasks and achieving goals. Agent systems are unique programmes of software systems which mainly adhere to the information domain [11]. They have the ability to react with a certain degree of independence, primarily for holding actions required for achieving their various goals [12].These agents are executed, especially when they are being operated in different environments.

(ii) *Agent-based load balancing*: An agent has the quality of performing particular tasks on its own, because of which it is known as an autonomous unit. The communication between agent and server usually does not take much time to process as compared to the centralised approach because of which the theme of agent-based load is readjusted and also minimises the servers' communication price [13]. Additionally, it enhances the pace of load balancing, which further increases the latency as well as the output of servers.

(iii) *Existing system*: The figure highlights the design of an existing system with several clients connected through the internet as a supplier of facilities that has revived many machines, systems of control, and the number of many mutual groups of resources that appear as clients [14]. One life cycle of an agent consists of the following steps:

   (i) *First step*: It acts as an preliminary server which gives commands to the final server for gathering data from all the servers that are actually needed for balancing load.
   (ii) *Second step*: Agent equals the ability of all the servers constructed to take the cloud's average load.

In the first step, an agent is started at an undefined server (mainly the initial server), which further finds several jobs in that server's line. After this, it calculates an average, depending on which, it will calculate the server's location and give a conclusion of whether it is underloaded or overloaded. The agent will then repeat this entire process till it reaches the last server. In the second step, the agent will reach from the initial server to the final server for the concerned load. On the particular server, it gives the solution for cross-checking the state. If the server is found to be overloaded, then the agent will transmit the jobs to those servers which are underloaded, and if the servers are underloaded, then it will get jobs from overloaded servers. The agents are known for doing this activity to the extent that the initial server will again adjust all the servers' load that contains the initial server. In this way, representatives are mainly utilised for balancing the load within the structure nodes.

3 Literature Review

Patel et al. [15] focused on distributing the remaining burden evenly across the compartment and shortening the make span. For load adjusting, various specialists have proposed various methodologies, such as genetic algorithm-based, PSO, ACO, Mn-Max, and so on. The authors of this paper has proposed a Grey-Wolf Optimization-based algorithm for adjusting load and shortening the make span. A GWO-based methodology was presented alongside GA and PSO-based algorithms.

Junaid et al. [16] proposed the ACOFTF half-breed algorithm, which takes into account significant QoS measurements such as SLA infringement, relocation time, throughput time, overhead time, and improvement time. Even in the presence of multiple datasets, the proposed model has demonstrated its ability to avoid untimely union, which was one of the goals of half-breed meta-heuristics. In the same way, a lack of variety encourages investigation, while a lot of variety often, but not always, leads to abuse.

Babou et al. [17] proposed the HEC Clustering Balance procedure, which fundamentally decreases the preparation time of client solicitations while productively utilising assets of the HEC node, MEC node, and the focal cloud on the three-level HEC engineering. The proposed strategy also allows for the use of HEC nodes that are not always selected. The proposed strategy uses a diagram to represent the cloud–MEC–HEC structure. Each MEC, known as a group head, is in charge of a sub-diagram, or bunch. There was a group head on each bunch (3-TER) that established the upper layer or 2-TER of the HEC design. The group heads are then linked together from one perspective and with the focal cloud from another.

Shahid et al. [18] focused on one of the CCMAN issues, such as load adjusting (LB). The goal of LB was to equilibrate the algorithm on the cloud nodes so that no host was under or overburdened. A few LB algorithms have been written in order to provide successful organisation and client fulfilment for fitting cloud nodes, to improve the general proficiency of cloud administrations, and to provide more fulfilment to the end client. A good LB algorithm improves productivity and resource use by dividing the remaining work among the framework's nodes in the best way possible.

Devraj et al. [19] proposed a load-adjusting algorithm based on the firefly and improved the multi-objective particle swarm optimization (MPSO) strategy, dubbed FMPSO. This method employs the firefly (FF) algorithm to narrow the search space, while the MPSO strategy was used to identify the improved response. The MPSO algorithm selects the global best (gbest) molecule with a small separation of highlights.

Saeedi et al. [20] present the asset skewness-mindful VMs' combination algorithm based on a more advanced thermodynamic recreated toughening strategy because asset skewness may prompt the algorithm to initiate extra workers. With the help of two heuristics and two meta-heuristics, the proposed SA-based algorithm has been tested in a wide range of situations with different amounts of skewness in the assets.

4 Proposed Work

4.1 Motivation

The existing system model agent-based dynamic load-balancing approach has a disadvantage, i.e. for every instance of a new job, the agent has to complete a life cycle (i.e. calculate the average and decide the status of the server). Hence, agents need to travel twice to allocate the job: once to find the average work load and a second one to balance the loads among nodes. To overcome this, we propose a multi-agent architecture for agent-based load balancing. A multi-agent architecture comprises Directory Agents and Mobile Agents. In this section, the proposed framework has been presented in detail. In the proposed work, we first provide a rank assignment framework to assign rank to the resources, and after that, an agent selection framework has been presented to select agent node. Finally, we present a resource allocation framework to allocate resources in an optimal manner to achieve high QoS and update the resource allocation table.

4.2 Rank Algorithm

Rc = 1 * initializing the resource counter*\
Rk = 1 * initializing resource rank*\
Cp = R[Rc]. Rpc * initializing the processing capacity of current resource to capacity of resource in the counter index*\
Cbw = R[Rc]. Rbw * initializing the bandwidth of current resource to bandwidth of resource in the counter index*\
Crt = R[Rc]. Rrt * initializing the response time of current resource to response time of resource in the counter index*\
Cl = R[Rc]. Rlc * initializing the latency of current resource to latency of resource in the counter index*\
For ∀ R[] **do** * repeat for all resources in the resource list*\
For i = 1:R[].size
If R[i]. Rpc ≥ Cp& R[i].Rbw ≥ Cbw&R[i]. Rrt ≥ Crt
&R[i]. Rlc ≥ Cl * compare the capacity of the current resource and the resource in the counter index*\
Cr = R[i] * set current resource to resource in the counter index if above condition is true*\
End for
Rank = Rk * rank the resource with the value in the rank variable*\
Update Cr Rank in At * update current resource in dynamic resource table*\
Remove(R[].Cc) *remove current resource from resource list*\
Rk = Rk + 1 * increment resource rank*\
Rc = Rc + 1 * increment resource counter*\

End for

In this algorithm, they assign rank to the resources. The rank of a resource is assigned based on parameters such as processing capacity, bandwidth, response time, and delay. The highest ranked resource capacity is updated for agent allocation in the agent information table.

After that, the host agent analyses the tasks and their requirements in the task list and chooses an appropriate layer for the completion of such tasks. Furthermore, once the connection is established, it is then up to the layer agent to perform the actual transmission.

4.3 Agent Selection Algorithm

Let Bmax, Pmax, Lcmax, Smax, and Rtmax represent bandwidth, processing capacity, latency, storage, and response time of the highest ranked resource in fog, high-capacity fog, and cloud layer.

Let Br, Pr, Lct, Sr, and Rtr represent task required bandwidth, processing capacity, task tolerable latency, task required storage, and response time.

The host agent selects an agent to allocate a resource for the execution of task by comparing the QoS requirement of the task and the capacity of the highest ranked resource in each host dynamic resource table.

For $\forall\ t\ \varepsilon$ R **do** * repeat for all task in a request*\
If $td \leq ac.\ Lcmax + ac.Rtmax\&Sr \leq ac.Smax$* compare task deadline with sum of latency & response time of cloud *\ Submit task to cloud agent * submit task to cloud agent if above condition is true*\
Else
Return (Resource unavailability message) * return failure due to resource unavailability*\
End if
End for

In this algorithm, the host agent selects an agent to allocate a resource for the execution of a task by comparing the QoS requirement of the task and the capacity of the highest ranked resource in each host dynamic resource table. An agent allocates a capable resource for a task received from a host agent by taking into account the deadline of the task, required processing capacity, bandwidth, response time, and tolerable latency. Any available resource that meets the requirements of the task is allocated for the execution of such a task.

4.4 Resource Allocation Algorithm

Let Bw, Pc, Lc, S, and Rt represent bandwidth, processing capacity, latency, storage, and response time of resource in fog, high-capacity fog, or cloud layer.

Let Br, Pr, Lct, Sr, Rtr, and td represent task required bandwidth, processing capacity, task tolerable latency, task required storage, response time, and task deadline.

The resource allocation algorithm selects the best available resource for execution of task that satisfies the QoS requirement of the task.

For ∀ t ε T[] * repeat for all task in task list*\
For ∀ r ε R[] * repeat for all resource in resource list*\
If td $\leq$ Lc + Rt& Pc $\geq$ Pr&r.status = free* compare task deadline with sum of latency & response time of resource *\
Allocate resource to task * allocate resource to task if above condition is true *\
r.status = allocated. * change resource status to allocated *\
t.status = allocated * change task status to allocated *\
End if
End for
If t.status = !Allocated * change task status not allocated *\
Return (Resource unavailability message) * return failure due to resource unavailability*\
End for

In this algorithm, the optimal resource for completing task is selected to fulfil the QoS of task.

4.5 Dynamic Resource Table Update Algorithm

Let R [], At, and Ct represent resource list, corresponding agent table, and current task executed by resource.

The dynamic resource table update algorithm updates the status of a resource after task execution is completed.

For ∀ r ε At. R[] * repeat for all resource in dynamic resource update table*\
tc = R[].Get (Ct) * get the current task executed by a resource*\
If tc = NULL * if task not found*\
r. status = free * change the status of the resource to free*\
End If
End for

Table 1 Simulation parameters

Parameters	Values
No of servers	100; 200; 300; 400
CTC (existing centralized scheme)	10 units
CTC (proposed)	1 unit
CTC (process of LB)	50 units
Platform	MATLAB 2013
Operating system	Windows 10

5 Results and Analysis

In this section, results and analysis have been presented. To implement the proposed mechanism, MATLAB is used. MATLAB is a general-purpose programming language developed by Mathworks. It provides a number of functions for different researchers to compute their simulations and plot results.

5.1 Performance Matrices

(i) *Throughput*: It indicates the total number of requests processed in perspective of total number of requests generated during data transmission.
(ii) *Latency*: It indicates the delay or time consumed by CPU during processing of requests by Agent.

5.2 Simulation Parameters

In this section, parameters used for simulation analysis have been discussed. To simulate a proposed mechanism, nodes are varied from 100 to 400. The simulation area is 100 × 100, and the initial energy of the node is 0.5Jule. Table 1 shows the complete parameters used for simulation.

5.3 Discussion

The simulation results of proposed and existing mechanisms are discussed in Figs. 1 and 2. In Fig. 1, a comparative analysis with the throughput parameter is presented, whereas Fig. 2 presents the latency of both proposed and existing techniques. The throughput is high in the proposed mechanism, whereas latency is less in the perspective of the existing single agent-based LB technique.

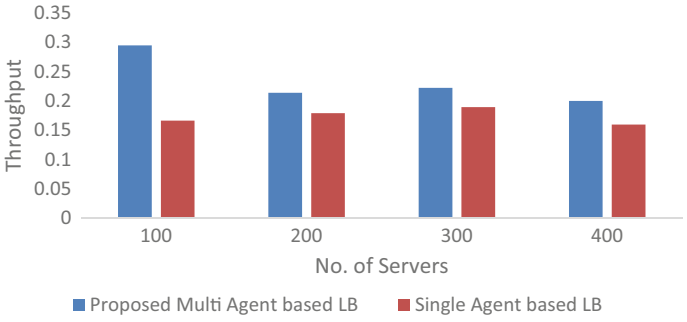

Fig. 1 Throughput of proposed Multi Agent_LB and Single Agent_LB

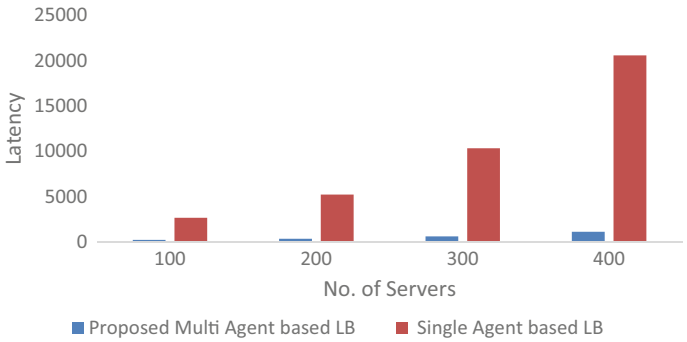

Fig. 2 Latency of proposed Multi Agent_LB and Single Agent_LB

6 Conclusion

The existing system model agent-based dynamic load-balancing approach has a disadvantage, i.e. for every instance of a new job, the agent has to complete a life cycle (i.e. calculate the average and decide the status of the server). Hence, agents need to travel twice to allocate the job: once to find the average work load and a second one to balance the loads among nodes. To overcome this, we propose a multi-agent architecture for agent-based load balancing. In this paper, an attempt has been made to propose a multi-agent-based load-balancing mechanism for cloud environments. A distributed load-balancing strategy based on multi-agent is proposed in this paper. The proposed mechanism has been simulated with the help of MATLAB. The simulation results show that throughput is high in the proposed mechanism, whereas latency is less in the perspective of the existing single agent-based LB technique. In the future, it is intended to continue working on it and apply the proposed mechanism in other environments, such as edge computing, which also improves throughput with less end-to-end delay.

References

1. Zhang J, Liu Q, Chen J (2016) A multi-agent based load balancing framework in cloud environment. In: 2016 9th international symposium on computational intelligence and design (ISCID), vol 1. IEEE, pp 278–281
2. George SS, Pramila RS (2021) A review of different techniques in cloud computing. Mater Today Proc 46:8002–8008
3. Shafiq DA, Jhanjhi NZ, Abdullah A, Alzain MA (2021) A load balancing algorithm for the data centres to optimize cloud computing applications. IEEE Access 9:41731–41744
4. Annie Poornima Princess G, Radhamani AS (2021). A hybrid meta-heuristic for optimal load balancing in cloud computing. J Grid Comput 19(2):1–22
5. Mohanty S, Patra PK, Ray M, Mohapatra S (2021) A novel meta-heuristic approach for load balancing in cloud computing. In: Research anthology on architectures, frameworks, and integration strategies for distributed and cloud computing. IGI Global, pp 504–526
6. Balaji K (2021) Load balancing in cloud computing: issues and challenges. Turk J Comput Math Educ 12(2):3077–3084
7. Kapila D, Dhir V (2021) Performance evaluation of new hybrid appraoch of load balancing in cloud computing. Des Eng 698–716
8. Jangra A, Mangla N, Jain A, Dewangan BK, Perumal T (2021) Classification of various scheduling approaches for resource management system in cloud computing. In: Autonomic computing in cloud resource management in industry 4.0. Springer, Cham, pp 149–157
9. Jangra A, Dubran H (2021) Simulation annealing based approach to enhanced load balancing in cloud computing. In: 2021 9th international conference on reliability. Infocom technologies and optimization (trends and future directions) (ICRITO). IEEE, pp 1–4
10. Mishra K, Majhi S (2020) A state-of-art on cloud load balancing algorithms. Int J Comput Digital Syst 9(2):201–220
11. Neelima P, Reddy A (2020) An efficient load balancing system using adaptive dragonfly algorithm in cloud computing. Clust Comput 23(4):2891–2899
12. Semmoud A, Hakem M, Benmammar B, Charr JC (2020) Load balancing in cloud computing environments based on adaptive starvation threshold. Concurr Comput Pract Exp 32(11):e5652
13. Jangra A, Mangla N (2021) Cloud LB using optimization techniques. In: Mobile radio communications and 5G networks. Springer, Singapore, pp 735–744
14. Singh N, Elamvazuthi I, Nallagownden P, Ramasamy G, Jangra A (2021) Assessment of microgrid communication network performance for medium-scale IEEE bus systems using multi-agent system. In: Mobile radio communications and 5G networks. Springer, Singapore, pp 377–387
15. Patel KD, Bhalodia TM (2019) An efficient dynamic load balancing algorithm for virtual machine in cloud computing. In: 2019 International conference on intelligent computing and control systems (ICCS). IEEE, pp 145–150
16. Junaid M, Sohail A, Rais RNB, Ahmed A, Khalid O, Khan IA, Huassin SS, Ejaz N (2020) Modeling an optimized approach for load balancing in cloud. IEEE Access 8:173208–173226
17. Babou CSM, Fall D, Kashihara S, Taenaka Y, Bhuyan MH, Niang I, Kadobayashi Y (2020) Hierarchical load balancing and clustering technique for home edge computing. IEEE Access 8:127593–127607
18. Shahid MA, Islam N, Alam MM, Su'ud MM, Musa S (2020) A comprehensive study of load balancing approaches in the cloud computing environment and a novel fault tolerance approach. IEEE Access 8:130500–130526
19. Devaraj AFS, Elhoseny M, Dhanasekaran S, Lydia EL, Shankar K (2020) Hybridization of firefly and improved multi-objective particle swarm optimization algorithm for energy efficient load balancing in cloud computing environments. J Parallel Distrib Comput 142:36–45
20. Saeedi P, Shirvani MH (2021) An improved thermodynamic simulated annealing-based approach for resource-skewness-aware and power-efficient virtual machine consolidation in cloud datacenters. Soft Comput 25(7):5233–5260