International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 2 Issue: 10 3038 – 3044

Load Balancing Techniques in Cloud Computing

Asitha Micheal
Department of Information Technology
Shah & Anchor Kutchhi Engineering College
Mumbai,India
asithamicheal@gamil.com

Jalpa Mehta
Department of Information Technology
line 2: name of organization, acronyms acceptable
Mumbai,India
jalpa03@yahoo.com

Abstract-As Cloud Computing is growing rapidly and clients are demanding more services and better results, load balancing for the Cloud has become a very interesting and important research area. The top challenges and Issues faced by cloud Computing is Security, Availability, Performance etc. The issue availability is mainly related to efficient load balancing, resource utilization & live migration of data in the server. In clouds, load balancing, as a method, is applied across different data centres to ensure the network availability by minimizing use of computer hardware, software failures and mitigating recourse limitations. Load Balancing is essential for efficient operations in distributed environments. Hence this paper presents the various existing load balancing Technique in Cloud Computing based on different parameters.

Keywords- *Cloud Computing, Distributed system, Dynamic Load balancing, Non-Distributed system*

*****

I. INTRODUCTION

Cloud computing is delivering software, Storage & infrastructure as a provisioned service to end users, but the underlying resource must be sufficiently scalable and robust. Whenever there is increase in demands Cloud vendors are based on automatic load balancing services, which allowed entities to increase the number of CPUs or memories for their resources to scale up according to their requirement. This service is optional and depends on the entity's business needs. Therefore load balancers served two important needs, primarily to promote availability of cloud resources and secondarily to promote performance.

Load Balancing is process of reassigning the total load to the individual nodes of the collective system to make resource utilization effective and to improve the response time of the job, simultaneously removing a condition in which some of the nodes are over loaded while some others are under loaded. Thus Load balancing is a relatively technique that facilitates networks and resources by providing a Maximum throughput with minimum response time by dividing the traffic between servers. Load balancing algorithms can be categorized mainly into two groups. They are Static and Dynamic load balancing.

II. STATIC LOAD BALANCING ALGORITHMS

Static Load balancing algorithms [1] assign the tasks to the nodes based only on the ability of the node to process new requests. The process is based solely on prior knowledge of the nodes' properties and capabilities. These would include the node's processing power, memory and storage capacity, and most recent known communication performance.

Although they may include knowledge of the communication prior performance, static algorithms generally do not consider dynamic changes of these attributes at run-time. In addition, these algorithms cannot adapt to load changes during run-time.

A. Round Robin and Randomized Algorithms

In the round robin [2], processes are divided evenly between all processors. Each new process is assigned to new processor in round robin order. The process allocation order is maintained on each processor locally independent of allocations from remote processors. With equal workload round robin algorithm is expected to work well. Round Robin and Randomized schemes work well with number of processes larger than number of processors. Advantage of Round Robin algorithm is that it does not require inter-process communication. Round Robin and Randomized algorithm both can attain the best performance among all load balancing algorithms for particular special purpose applications. In general Round Robin and Randomized are not expected to achieve good performance in general case.

B. Central Manager Algorithm

In this algorithm [2], a central processor selects the host for new process. The minimally loaded processor depending on the overall load is selected when process is created. Load manager selects hosts for new processes so that the processor load confirms to same level as much as possible. From that information on the system load state, central load manager makes the load balancing judgment. This information is updated by remote processors, which send a message each time to the load manager on changes. This information can depend on waiting of parent's process of completion of its children's process, end of parallel execution. The load manager makes load balancing decisions based on the system load information, allowing the best decision when of the process created. High degree of inter-process communication could make the bottleneck state. This algorithm is expected to perform better than the parallel applications, especially when dynamic activities are created by different hosts.

C. Threshold Algorithm

According to this algorithm [2], the processes are assigned immediately upon creation to hosts. Hosts for new processes are selected locally without sending remote messages. Each processor keeps a private copy of the system's load. The load of a processor can characterize by one of the three levels: under loaded, medium and overloaded. Two threshold parameters tunder and tupper can be used to describe these levels.

Under loaded - load < tunder
Medium - tunder ≤ load ≤ tupper
Overloaded - load > tupper

Initially, all the processors are considered to be under loaded. When the load state of a processor exceeds a load level limit, then it sends messages regarding the new load state to all remote processors, regularly updating them as to the actual load state of the entire system.

If the local state is not overloaded then the process is allocated locally. Otherwise, a remote under loaded processor is selected, and if no such host exists, the process is also allocated locally. Thresholds algorithm have low inter process communication and a large number of local process allocations. The later decreases the overhead of remote process allocations and the overhead of remote memory accesses, which leads to improvement in performance.

III. DYNAMIC LOAD BALANCING ALGORITHMS

Dynamic load balancing algorithms [1] take into account the different attributes of the nodes' capabilities and network bandwidth. Most of these algorithms rely on a combination of Knowledge based on prior gathered information about the nodes in the Cloud and run-time properties collected as the selected nodes process the task's components. These algorithms assign the tasks and may dynamically reassign them to the nodes based on the attributes gathered and calculated. Such algorithms require constant monitoring of the nodes and task progress and are usually harder to implement. However, they are more accurate and could result in more efficient load balancing. Dynamic load balancing can be done in two different ways: distributed and non-distributed.

A. Distributed system

In the distributed one, the dynamic load balancing algorithm is executed by all nodes present in the system and the task of load balancing is shared among them. The interaction among nodes to achieve load balancing can take two forms: cooperative and non-cooperative. In the first one, the nodes work side-by-side to achieve a common objective, for example, to improve the overall response time, etc. In the second form, each node works independently toward a goal local to it, for example, to improve the response time of a local task. Dynamic load balancing algorithms of distributed nature, usually generate more messages than the non-distributed ones because, each of the nodes in the system needs to interact with every other node. A benefit, of this is that even if one or more nodes in the system fail, it will not cause the total load balancing process to halt; it instead would affect the system performance to some extent.

B. Non-Distributed System

In non-distributed type, either one node or a group of nodes do the task of load balancing. Non-distributed dynamic load balancing algorithms can take two forms: centralized and semi-distributed. In the first form, the load balancing algorithm is executed only by a single node in the whole system: the central node. This node is solely responsible for load balancing of the whole system. The other nodes interact only with the central node. In semi-distributed form, nodes of the system are partitioned into clusters, where the load balancing in each cluster is of centralized form. A central node is elected in each cluster by appropriate election technique which takes care of load balancing within that cluster. Hence, the load balancing of the whole system is done via the central nodes of each cluster. Centralized dynamic load balancing takes fewer messages to reach a decision, as the number of overall interactions in the system decreases drastically as compared to the semi-distributed case. However, centralized algorithms can cause a bottleneck in the system at the central node and also the load balancing process is rendered useless once the central node crashes. Therefore, this algorithm is most suited for networks with small size.

1) Central Queue Algorithm

Central Queue Algorithm [2] works on the principle of dynamic distribution. It stores new activities and unfulfilled requests as a cyclic FIFO queue on the main host. Each new activity arriving at the queue manager is inserted into the queue. Then, whenever a request for an activity is received by the queue manager, it removes the first activity from the queue and sends it to the requester. If there are no ready activities in the queue, the request is buffered, until a new activity is available. If a new activity arrives at the queue manager while there are unanswered requests in the queue, the first such request is removed from the queue and the new activity is assigned to it. When a processor load falls under the threshold, the local load manager sends a request for a new activity to the central load manager. The central load manager answers the request immediately if a ready activity is found in the process-request queue, or queues the request until a new activity arrives.

2) Local Queue Algorithm

Main feature of this algorithm is dynamic process migration support. The basic idea of the local queue algorithm [2] is static allocation of all new processes with process migration initiated by a host when its load falls under threshold limit, as a user-defined parameter of the algorithm. The parameter defines the minimal number of ready processes the load manager attempts to provide on each processor. Initially, new processes created on the main host are allocated on all under loaded hosts. The number of parallel activities created by the first parallel construct on the main host is usually sufficient for allocation on all remote hosts. From then on, all the processes created on the main host and all other hosts are allocated locally. When the host gets under loaded, the local load manager attempts to get several processes from remote hosts. It randomly sends requests with the number of local ready processes to remote load managers. When a load manager receives such a request, it compares the local number of ready processes with the received number. If the former is greater than the latter, then some of the running processes are transferred to the requester and an affirmative confirmation with the number of processes transferred is returned.

3) Honeybee Foraging Algorithm

This algorithm [3] is derived from the behavior of honey bees for finding and reaping food. There is a class of bees called the forager bees which forage for food sources, upon finding one, they come back to the beehive to advertise this using a dance called waggle dance. The display of this dance, gives the idea of the quality or quantity of food and also its distance from the beehive. Scout bees then follow the foragers to the location of food and then began to reap it. They then return to the beehive

and do a waggle dance, which gives an idea of how much food is left and hence results in more exploitation or abandonment of the food source.

In case of load balancing, as the webservers demand increases or decreases, the services are assigned dynamically to regulate the changing demands of the user. The servers are grouped under virtual servers (VS), each VS having its own virtual service queues. Each server processing a request from its queue calculates a profit or reward, which is analogous to the *quality that the bees show in their waggle dance. One measure* of this reward can be the amount of time that the CPU spends on the processing of a request. The dance floor in case of honey bees is analogous to an advert board here. This board is also used to advertise the profit of the entire colony.

Each of the servers takes the role of either a forager or a scout. The server after processing a request can post their profit on the advert boards with a probability of pr. A server can choose a queue of a VS by a probability of px showing forage/explore behavior, or it can check for advertisements (see dance) and serve it, thus showing scout behavior. A server serving a request, calculates its profit and compare it with the colony profit and then sets its px. If this profit was high, then the server stays at the current virtual server; posting an advertisement for it by probability pr. If it was low, then the server returns to the forage or scout behavior.

4) Biased Random Sampling

Here a virtual graph is constructed, with the connectivity of each node (a server is treated as a node) representing the load on the server. Each server is symbolized as a node in the graph, with each in degree directed to the free resources of the server. Regarding job execution and completion,

- Whenever a node does or executes a job, it deletes an incoming edge, which indicates reduction in the availability of free resource.
- After completion of a job, the node creates an incoming edge, which indicates an increase in the availability of free resource.

The addition and deletion of processes is done by the process of random sampling. The walk starts at any one node and at every step a neighbor is chosen randomly. The last node is selected for allocation for load. Alternatively, another method can be used for selection of a node for load allocation, that being selecting a node based on certain criteria like computing efficiency, etc. Yet another method can be selecting that node for load allocation which is under loaded i.e. having highest in degree. If b is the walk length, then, as b increases, the efficiency of load allocation increases. This defines a threshold value of b, which is generally equal to log n experimentally.

A node upon receiving a job, will execute it only if its current walk length is equal to or greater than the threshold value. Else, the walk length of the job under consideration is incremented and another neighbor node is selected randomly. When, a job is executed by a node then in the graph, an incoming edge of that node is deleted. After completion of the job, an edge is created from the node initiating the load allocation process to the node which was executing the job.

Finally a directed graph is achieved. The load balancing scheme used here is fully decentralized, thus making it apt for large network systems like that in a cloud [3].

5) Active Clustering

Active Clustering [3] is considered in as a self-aggregation algorithm to rewire the network. This algorithm works on the principle of grouping similar nodes together and working on these groups. Many load balancing algorithms only work well where the nodes are aware of "like" nodes and can delegate workload to them. The process involved is:

- A node initiates the process and selects another node called the matchmaker node from its neighbours satisfying the criteria that it should be of a different type than the former one.
- The so called matchmaker node then forms a connection between neighbors of it which is of the same type as the initial node.
- The matchmaker node then detaches the connection between itself and the initial node.

The above set of processes is followed iteratively.

IV. EXISTING LOAD BALANCING TECHNIQUES IN CLOUDS

Following load balancing techniques are currently prevalent in clouds

A. Decentralized content aware

H. Mehta et al. [4] proposed a new content aware load balancing policy named as work-load and client aware policy (WCAP). It uses a parameter named as USP to specify the unique and special property of the requests as well as computing nodes. USP helps the scheduler to decide the best suitable node for processing the requests. This strategy is implemented in a decentralized manner with low overhead. By using the content information to narrow down the search, it improves the searching performance overall performance of the system. It also helps in reducing the idle time of the computing nodes hence improving their utilization.

B. CARTON

R. Stanojevic et al. [4] proposed a mechanism CARTON for cloud control that unifies the use of LB and DRL. LB (Load Balancing) is used to equally distribute the jobs to different servers so that the associated costs can be minimized and DRL (Distributed Rate Limiting) is used to make sure that the resources are distributed in a way to keep a fair resource allocation. DRL also adapts to server capacities for the dynamic workloads so that performance levels at all servers are equal. With very low computation and communication overhead, this algorithm is simple and easy to implement.

C. Compare and Balance

Y. Zhao et al. [4] addressed the problem of intra-cloud load balancing amongst physical hosts by adaptive live migration of virtual machines. A load balancing model is designed and implemented to reduce virtual machines' migration time by shared storage, to balance load amongst servers according to their processor or IO usage, etc. and to keep virtual machines' zero-downtime in the process. A distributed load balancing algorithm COMPARE AND BAL-ANCE is also proposed that is based on sampling and reaches equilibrium very fast. This algorithm assures that the migration of VMs is always from high-cost physical hosts to low-cost host but assumes that each

physical host has enough memory which is a weak assumption.

D. Event-driven

V. Nae et al. [4] presented an event-driven load balancing algorithm for real-time Massively Multiplayer Online Games (MMOG). This algorithm after receiving capacity events as input, analyzes its components in context of the resources and the global state of the game session, thereby generating the game session load balancing actions. It is capable of scaling up and down a game session on multiple resources according to the variable user load but has occasional QoS breaches.

E. Scheduling strategy on LB of VM resources [SS on LB of VM resource]

J. Hu et al. [4] proposed a scheduling strategy on load balancing of VM resources that uses historical data and current state of the system. This strategy achieves the best load balancing and reduced dynamic migration by using a genetic algorithm. It helps in resolving the issue of load imbalance and high cost of migration thus achieving better resource utilization.

F. CLBVM

A. Bhadani et al. [4] proposed a Central Load Balancing Policy for Virtual Machines (CLBVM) that balances the load evenly in a distributed virtual machine/cloud computing environment. This policy improves the overall performance of the system but does not consider the systems that are fault-tolerant.

G. LBVS

H. Li. [4] proposed a load balancing virtual storage strategy (LBVS) that provides a large scale net data storage model and Storage as a Service model based on Cloud Storage. Storage virtualization is achieved using an architecture that is three-layered and load balancing is achieved using two load balancing modules. It helps in improving the efficiency of concurrent access by using replica balancing further reducing the response time and enhancing the capacity of disaster recovery. This strategy also helps in improving the use rate of storage resource, flexibility and robustness of the system.

H. Task Scheduling based on LB

Y. Fang et al. [4] discussed a two-level task scheduling mechanism based on load balancing to meet dynamic requirements of users and obtain high resource utilization. It achieves load balancing by first map-ping tasks to virtual machines and then virtual machines to host resources thereby improving the task response time, resource utilization and overall performance of the cloud computing environment.

I. Honeybee Foraging Behavior

M. Randles et al. [4] investigated a decentralized honeybee-based load balancing technique that is a nature-inspired algorithm for self-organization. It achieves global load balancing through local server actions. Performance of the system is enhanced with increased sys-tem diversity but throughput is not increased with an increase in system size. It is best suited for the conditions where the diverse population of service types is required.

J. Biased Random Sampling

M. Randles et al. [4] investigated a distributed and scalable load balancing approach that uses random sampling of the system domain to achieve self-organization thus balancing the load across all nodes of the system. The performance of the system is improved with high and similar population of resources thus resulting in an in-creased throughput by effectively utilizing the increased sys-tem resources. It is degraded with an increase in population diversity.

K. Active Clustering

M. Randles et al. [4] investigated a self-aggregation load balancing technique that is a self-aggregation algorithm to optimize job assignments by connecting similar services using local re-wiring. The performance of the system is enhanced with high resources thereby in-creasing the throughput by using these resources effectively. It is degraded with an increase in system diversity.

L. ACCLB

Z. Zhang et al. [4] proposed a load balancing mechanism based on ant colony and complex network theory (ACCLB) in an open cloud computing federation. It uses small-world and scale-free characteristics of a complex network to achieve better load balancing. This technique overcomes heterogeneity, is adaptive to dynamic environments, is excel-lent in fault tolerance and has good scalability hence helps in improving the performance of the system.

M. (OLB + LBMM)

S.-C. Wang et al. [4] proposed a two-phase scheduling algorithm that combines OLB (Opportunistic Load Balancing) and LBMM (Load Balance Min-Min) scheduling algorithms to utilize better executing efficiency and maintain the load balancing of the system. OLB scheduling algorithm, keeps every node in working state to achieve the goal of load balance and LBMM scheduling algorithm is utilized to minimize the execution time of each task on the node thereby minimizing the overall completion time. This combined approach hence helps in an efficient utilization of resources and enhances the work efficiency.

N. VectorDot

A. Singh et al. [4] proposed a novel load balancing algorithm called VectorDot. It handles the hierarchical complexity of the data-center and multidimensionality of resource loads across servers, network switches, and storage in an agile data center that has integrated server and storage virtualization technologies. VectorDot uses dot product to distinguish nodes based on the item requirements and helps in removing overloads on servers, switches and storage nodes.

O. Server-based LB for Internet distributed services [Server Based LB for IDS]

A. M. Nakai et al. [4] proposed a new server-based load balancing policy for web servers which are distributed all over the world. It helps in reducing the service response times by using a protocol that limits the redirection of requests to the closest remote servers without overloading them. A middleware is described to implement this protocol. It also uses a heuristic to help web servers to endure overloads.

P. *Join-Idle-Queue*

Y. Lua et al. [4] proposed a Join-Idle-Queue load balancing algorithm for dynamically scalable web services. This algorithm provides large-scale load balancing with distributed dispatchers by, first load balancing idle processors across dispatchers for the availability of idle processors at each dispatcher and then, assigning jobs to processors to reduce average queue length at each processor. By removing the load balancing work from the critical path of request processing, it effectively reduces the system load, incurs no communication overhead at job arrivals and does not increase actual response time.

Q. *Lock-free multiprocessing solution for LB [LF multiprocessing solution for LB]*

X. Liu et al. [4] proposed a lock-free multiprocessing load balancing solution that avoids the use of shared memory in contrast to other multiprocessing load balancing solutions which use shared memory and lock to maintain a user session. It is achieved by modifying Linux kernel. This solution helps in improving the overall performance of load balancer in a multi-core environment by running multiple load-balancing processes in one load balancer.

V. METRICS FOR LOAD BALANCING IN CLOUDS [5]

The existing load balancing techniques in clouds consider various parameters like performance, response time, scalability, throughput, resource utilization, fault tolerance, migration time and associated overhead. But, for an energy-efficient load balancing, metrics like energy consumption and carbon emission should also be considered.

A. *Overhead Associated*

Overhead Associated determines the amount of overhead involved while implementing a load-balancing algorithm. It is composed of overhead due to movement of tasks, inter-processor and inter-process communication. This should be minimized so that a load balancing technique can work efficiently.

B. *Throughput*

Throughput is used to calculate the no. of tasks whose execution has been completed. It should be high to improve the performance of the system

C. *Performance*

It is used to check the efficiency of the system. It has to be improved at a reasonable cost e.g. reduce response time while keeping acceptable delays.

D. *Resource Utilization*

Resource Utilization is used to check the utilization of resources. It should be optimized for an efficient load balancing.

E. *Scalability*

Scalability is the ability of an algorithm to perform load balancing for a system with any finite number of nodes. This metric should be improved.

F. *Response Time*

Response Time is the amount of time taken to respond by a particular load balancing algorithm in a distributed system. This parameter should be minimized.

G. *Fault Tolerance*

Fault Tolerance is the ability of an algorithm to perform uniform load balancing in spite of arbitrary node or link failure. The load balancing should be a good fault-tolerant technique.

H. *Migration time*

It is the time to migrate the jobs or resources from one node to other. It should be minimized in order to enhance the performance of the system.

I. *Energy Consumption (EC)*

EC determines the energy consumption of all the resources in the system. Load balancing helps in avoiding overheating by balancing the workload across all the nodes of a Cloud, hence reducing energy consumption.

J. *Carbon Emission (CE)*

CE calculates the carbon emission of all the resources in the system. As energy consumption and carbon emission go hand in hand, the more the energy consumed, higher is the carbon footprint. So, for an energy-efficient load balancing solution, it should be reduced.

Based on the above metrics, the existing load balancing techniques have been compared in Table 1

TABLE 1: METRICS CONSIDERED BY EXISTING LOAD BALANCING TECHNIQUES [4]

TECHNIQUES	PERFORMANCE	RESPONSE	SCALABILITY	OVERHEAD	THROUGHPUT	RESOURCE UTILIZATION	FAULT TOLERANCE	MIGRATION TIME	EC	CE
DECENTRALIZED CONTENT AWARE	✓	✓	✓	✓	×	✓	×	×	×	×
CARTON	✓	×	×	✓	×	✓	×	×	×	×
COMPARE & BALANCE	×	×	×	✓	×	✓	×	✓	×	×
EVENT-DRIVEN	×	×	✓	×	×	✓	×	×	×	×
SS ON LB OF VM	×	×	×	✓	×	✓	×	×	×	×
CLBVM	✓	✓	×	×	✓	✓	×	×	×	×
LBVS	✓	✓	✓	×	×	×	✓	×	×	×
TASK SCHEDULING BASED ON LB	✓	✓	×	×	×	✓	×	×	×	×
HONEY BEE FORAGING BEHAVIOUR	✓	×	✓	×	✓	×	×	×	×	×
BIASED RANDOM SAMPLING	✓	×	✓	×	✓	×	×	×	×	×
ACTIVE CLUSTERING	✓	×	✓	×	✓	×	×	×	×	×
ACCLB	✓	×	×	×	×	×	✓	×	×	×
OLB+LBMM	×	×	×	×	×	×	✓	×	×	×
VECTOR DOT	×	×	×	×	×	×	✓	×	×	×
SERVER BASED LB FOR IDS	✓	✓	×	×	×	×	×	×	×	×
JOIN-IDLE QUEUE	✓	✓	×	✓	×	×	×	×	×	×
LF MULTIPROCESSING SOLUTION FOR LB	✓	×	×	×	✓	×	×	×	×	×

VI. CONCLUSION

This paper presented different load balancing techniques of Cloud Computing and comparative analysis between them. It also surveyed multiple load balancing algorithms. The proposed algorithms were based on one or more techniques. It was noted that there are no algorithm which can achieve all the metrics of Load Balancing. Each and every algorithm was designed to achieve specific objective, such as Biased Random Sampling algorithm has efficient performance & scalability but poor Fault tolerance. Therefore, such algorithm must be designed which can handle different types of workload and suitable for all types of enviorment.

REFERENCES

[1] Klaithem Al Nuaimi, Nader Mohamed, Mariam Al Nuaimi and Jameela Al-Jaroodi "A Survey of Load Balancing in Cloud Computing:Challenges and Algorithms" 2012 IEEE Second Symposium on Network Cloud Computing and Applications

[2] Sandeep Sharma, Sarabjit Singh, and Meenakshi Sharma, "Performance Analysis of Load Balancing Algorithms" World Academy of Science, Engineering and Technology 14 2008

[3] Ram Prasad Padhy and P Goutam Prasad Rao, "Load Balancing In Cloud Computing Systems" Thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Technology NIT Rourkela May, 2011

[4] Nidhi Jain Kansal, Inderveer Chana "Cloud Load Balancing Techniques : A Step Towards Green Computing " IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 1, No 1, January 2012

[5] Nidhi Jain Kansal* and Inderveer Chana , "Existing Load Balancing Techniques In Cloud Computing: A Systematic Re-View" ISSN: 0976-8742, E-ISSN: 0976-8750, Volume 3, Issue 1, 2012