Comparative Study of Various Load Balancing Algorithms

Yash Verma[1], Manvendra Singh Chhajerh[2]

[1]Student, Dept. of Information Science and Engineering, R.V. College of Engineering, Karnataka, India
[2]Student, Dept. of Information Science and Engineering, R.V. College of Engineering, Karnataka, India

---

Abstract - *In today's world all the customers want instantaneous answers to their queries or services that they've asked for and every other company is racing against time to serve the customer request as soon as possible. Companies have a set of servers to answer their queries but the problem is how to assign these requests to servers in the most optimized way. If we assign all the requests to one server then it might crash and others might sit idle, hence to overcome all such problems we need a balancer to balance the load in an optimized way and this is done by a load balancer. Finding a better algorithm for a load balancer is becoming a hot topic of research and therefore a lot of such algorithms are available. This paper evaluates and compares some most important algorithms for load balancing.*

Key Words: **Load Balancing, Static Load Balancing, Dynamic Load Balancing, Round Robin, MMLB, Consistent Hashing, Join-the-Shortest-Queue, Join-the-Idle-Queue,Power of d choices**

1.INTRODUCTION

To provide a good user experience, the companies need to reply back to customer actions on their website, online applications, etc. and these actions or we say them as client requests need to be served by a server. To balance the traffic of these requests among all the servers a load balancer is used. Load balancing refers to managing and distributing these client requests among two or more servers. This is essential to make sure that no server is overloaded with requests and requests are distributed evenly such that the overall responsive time for a customer is minimum.

Load balancing does not help in improving the performance of a cluster of server but also helps when the company wants to scale their business by adding more servers or when a company's server is down due to any reason, may it be power failure or due to network card failure, load balancing makes sure that the request of that server is re-assigned to some another server. Some algorithms are very easy to implement while other are very complex, some provides high response time while others provide low overhead time, some has high throughput while others are easy to scale.

Thus arises a need to analyse these algorithms based on such factors, so that suitability of these algorithms under various conditions can be established.

2. Load Balancing Techniques

Load balancing decisions taken by the load balancer are driven through a number of factors. One of the most important factor is the state of the system. Based on this, whether the load balancing decision involves the state of the system or not, load balancing techniques are classified into two categories, namely Static load balancing and Dynamic Load Balancing. Static load balancing does not account for the current state of the system while making load balancing decision, while dynamic load balancing considers the current state of the system as a crucial factor and distributes the load among server accordingly. [1]

2.1 Static Load Balancing

Static Load Balancing relies on a beforehand knowledge of the application and uses this information to distribute the load uniformly. It uses statistical information about the system for load distribution among the servers. Static Load balancing has low queuing delays as the the decision is not based upon the current state of the system rather decided using a set of rules independent of the state. The processing power and speed of the servers are decided at the start and remains invariable throughout [2]. Static Load balancing has its shortcomings also. Since the current state of the system is not considered and the processor performance is also decided beforehand, the decision takens using static load balancing can result in unbalanced load distribution and result in overutilization of some servers while other remains idle.[3]

2.1 Dynamic Load Balancing

Dynamic load balancing algorithms follow an approach to forward the incoming task to the lightest server among all the servers. The decision of determination of the lightest server is the main task of any dynamic laid balancing server. Dynamic Load balancing algorithms make smart decision based on the current state of the system to equally distribute load. The processes can be shifted to server with low load from a server at high load at runtime. The Complexity of these algorithms are relatively higher than static load balancing algorithms, but they provide much better fault-tolerance and performance than their counterpart. Dynamic Load balancing algorithms are much preferred for systems where behavior of the load can be predicted before and the variations are high, while static load balancing algorithms are preferred choice when the load is light and variations are less.

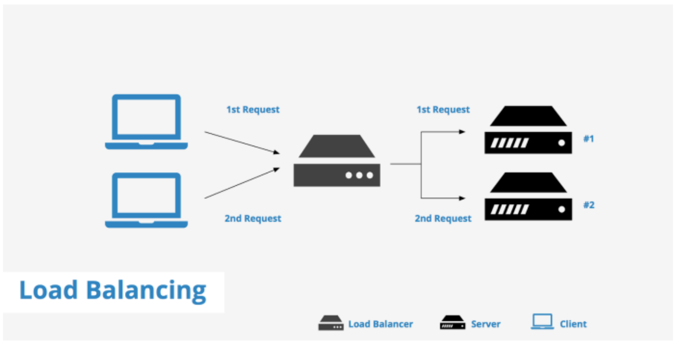

Fig -1: Load Balancing Model

3. Load Balancing Techniques

3.1 Round Robin

It is a very simple algorithm that balances the load by distributing the client request equally among the servers. It iterates over all the servers in a circular fashion i.e., first it distributes some defined load to servers one by one and once it encounters the last server it goes back to the first server. For example: If there are 3 servers, say X,Y & Z, then with this algorithm the load balancer sends 1st client request to X, then 2nd client request to Y ,then 3rd request to Z , then 4th client request back to X and the cycle goes on.
Characteristics:
Implementation: This algorithm is simplest to implement.
Performance: It works good in an ideal environment but in real life where the capacity of servers may differ, and any server may go down it's not good.
Throughput: is moderate in Round Robin
Scalability: Easy to scale. High

3.2 Weighted Round Robin

This algorithm is an efficient version of round robin. This algorithm accounts for differences in efficiency or power of servers. In this, each server is assigned some weight based on its working capacity, higher the capacity more the weight. Based on weight assigned, the servers are given a number of client requests. This helps in better resource utilization. This also leads to increase in throughput as compared to round robin.

3.3 Opportunistic Load Balancing

It is a static load balancing algorithm that randomly distributes the client requests to different servers. It doesn't account the present workload on the servers and distributes randomly, which results in slow execution of the request and since it hasn't calculated the implementation time it sometimes results in bottlenecks in spite of having some free servers. All this leads to poor performance of load balancing and therefore this algorithm is not used directly rather in a hybrid way.

3.4 Min-Min Load Balancing

This algorithm is simple and effective. It first calculates the execution time of all the requests and finds the minimum execution time request. Based on this minimum execution time it assigns this task to the server with minimum total execution time. It then updates the total execution time of that server by increasing it with the execution time of the request assigned. This process continues till all the requests are assigned to one or the other server.

3.5 Consistent Hashing

It is a static load balancing algorithm that distributes the incoming requests based on some mathematical computation known as hashing. The algorithm assigns some unique values to the servers based on their name or ID, and then assigns the incoming requests to the server with the nearest hash value. The hash value of the incoming request is calculated at the load balancer by using a mathematical function known as hashing function. The incoming request can be hashed for, or a combination of - Source IP, Source IP and port, Uniform Resource Locator, Header value

Consistent Hashing is favorable for handling a dynamic number of servers as it provides persistence in a way that addition or removal of servers does not result in recalculation of hash tables. But the major limitation of this algorithm owing to the mathematical properties, comes when a large number of requests are targeted for a specific service, the consistent hashing algorithm will transfer the request to the same subset of servers resulting in imbalance of load. The algorithm is much more efficient at handling uniform requests to a number of servers.[4][5][6]

3.6 Consistent Hashing with Bounder Load

Consistent hashing with bounded load is an upgraded version of consistent hashing that tackles the imbalance that occurs due to more requests for a specific server. This algorithm introduces a constant 'c' called as balancing factor whose value can go from 1 to infinity. At value 1 the algorithm behaves as least connection algorithm while when c tends to infinity it behaves as simple consistent hashing. The balancing factor 'c' defines an upper bound on the load that the server can accommodate. As an example. at a value of c as 1.1 the maximum load that this server can handle is 110% of the average load of the system. If the server is overloaded meaning the capacity is more than c times, the requests get transferred to the nearest server with value closest to the hash value of request. If the server is not overloaded, then it behaves like a consistent hashing algorithm.[7]

3.7 Join-the-shortest-queue

Join the Shortest Queue (JSQ) algorithm is based on a greedy policy. The environment consists of a single dispatcher or load balancer and a number of servers. All the incoming requests are handled by the dispatcher sent to a particular server and the responses from the server are again handled by the dispatcher. The dispatcher upon receiving a request from the network, transfers it to the server with the shortest queue i.e. with least number of jobs. This approach is greedy from the perspective of the incoming request in the way that the incoming request gets transferred to the server where it can be processed fastest owing to the least number of jobs. Since every request and response is handled by the dispatcher, a communication overhead is involved since JSQ requires the instantaneous length of the queue at each server. The limitation comes with JSQ when a central dispatcher is replaced by a distributed design to serve a very large number of servers to avoid bottlenecks at the dispatcher. Communication Overhead is increased significantly resulting in traffic at critical path for request processing which in turn reduce response time.[8]

3.8 Join-the-idle-queue

Join the Idle Queue (JIQ) algorithm is proposed for a system of dynamically scalable web servers with distributed design of load balancers. The distributed design of load balancers proposes a challenge as each balancer tries to balance the load independently of other dispatchers and since each dispatcher processes a small part of the overall incoming request it does not have the full knowledge about the number of jobs at each server. The JIQ algorithm overcomes this challenge by introducing a term called secondary load balancing. Primary load balancing involves distribution of incoming requests to the servers while secondary load balancing involves informing the load balancers about the idleness of the server. Secondary load balancing occurs in the reverse direction irrespective of the job arrival requests while results in offloading of the critical path of request processing. Since in a distributed server farm it is challenging to determine which dispatcher to select to inform, the algorithm uses two techniques viz Random sampling and the Power of d (SQ(d)) algorithm. The JIQ-Random technique selects a dispatcher queue uniformly at random while JIQ-SQ(d) selects d dispatchers at random and sends the information to the dispatchers with least number of informing queue length.[9]

3.9 Power of D choices

Power of d scheme is a generalized scheme of load balancing algorithm. It assumes a system of N servers with a single dispatcher that balances the load among the N servers. In this, the dispatcher selects a random d number of servers from N servers and transfers the requested to the one with the shortest queue length similar in the way JSQ operates. It is generalized on top of the JSQ algorithm and is given as the term JSQ(d(N)) where d(N) represents a randomly selected subset of d servers from N servers. The Join-the-shortest queue and Join-the-idle queue are considered to be special cases for the power of d scheme when assuming d(N) = N and d(N) =1 respectively. The Power of D improves the scalability of JSQ. Power of d choices are much more effective in balancing load (balanced queue lengths) than the algorithms that opt for a randomized selection of servers but JSQ schemes outperforms JSQ(d(N)) in terms of delay for reasons that JSQ has about all N servers and can select optimally among those while JSQ(d(N)) can only at a time select among d servers.[10][11]

Comparison

On the analysis of various load balancing algorithms few metrics emerge out to basis of comparison among these algorithms. We define these metrics as –

Throughput - throughput is a performance metric that indicates the number of jobs that have successfully completed their execution on the servers. A High throughput value for a particular algorithm is preferred.

Response Time - response time can be defined as the time interval between the arrival of the job on load balancer and its subsequent forwarding to one of the servers. The value of this metric should be low.

Overhead Communication - This involves the extra effort or work that is required to achieve the load balancing output. It can alternatively be defined as the number of transactions between the load balancer and servers to determine the forwarding of a single incoming packet. This parameter should be kept low to improve the overall efficiency of the system.

Complexity of Implementation - This parameter describes the complexity of functioning of load balancing algorithms. An algorithm with low complexity of implementation is preferred.

Scalability - Scalability of the system can be defined as the ability of the system to increase the number of servers as the load increases. A high scalability is desired for load balancers to manage the increasing load.

Resource Utilization - This metric is defined as the total load on the system by the total working capacity of the system. High value of resource utilization is preferred to increase the value invested in the system.

Table -1: Comparison of load balancing algorithms

Parameters / Algorithms	Type	Throughput	Response Time	Overhead Communication	Complexity	Scalability	Resource Utilization
Round Robin	S	M	L	L	E	H	L
Weighted R R	S	M	L	L	E	H	M
OLB	S	L	L	L	E	H	L
MMLB	S	M	M	M	C	M	H
CH	S	H	M	L	H	H	L
CHBL	D	H	L	L	H	H	L
JIQ	D	H	L	M	M	M	H
JSQ	D	H	H	H	M	L	H
Power of d	D	H	L	H	M	H	M

4. CONCLUSIONS

In today's world, where the internet plays a very important role, the number of users are increasing at a very high rate and to serve those requests efficiently among servers, we need load balancers. This paper gives a brief introduction about what is load balancing and its importance, static load balancing algorithms, dynamic load balancing algorithms, different types of algorithms followed by metrics on which we can compare these algorithms. This paper highlights the pros and cons of various load balancing algorithms. This paper shows a comparative analysis of various load balancing algorithms, which can help in building a hybrid algorithm to counter the problems of the algorithm.

REFERENCES

[1] N. J. Kansal and I. Chana, "Existing Load Balancing Techniques in Cloud Computing: A Systematic Review," vol. 3, issue. 1, pp. 87-91, 2012.

[2] Y. Sahu and R. K. Pateriya, "Cloud Computing Overview with Load Balancing Techniques," vol. 65, no. 24, pp. 40-44, 2013.

[3] Nadeem Shah, Mohammed Farik, "Static Load Balancing Algorithms In Cloud Computing: Challenges & Solutions" ,International Journal of Scientific & Technology Research, Vol 4, pp 353-355

[4] Load Balancing Algorithms. (n.d.). Retrieved May 18, 2021, from https://avinetworks.com/docs/17.2/load-balancing-algorithms/#consistent-hash

[5] Karger, D. R., & Ruhl, M. (2004). *Simple efficient load balancing algorithms for peer-to-peer systems. Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures - SPAA '04.* doi:10.1145/1007912.1007919

[6] Li, J., Nie, Y., & Zhou, S. (2018). *A Dynamic Load Balancing Algorithm Based on Consistent Hash. 2018 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC).* doi:10.1109/imcec.2018.8469341

[7] Mirrokni, V., Thorup, M., & Zadimoghaddam, M. (2018). *Consistent Hashing with Bounded Loads. Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 587–604.* doi:10.1137/1.9781611975031.39

[8] Gupta, V., Harchol Balter, M., Sigman, K., & Whitt, W. (2007). *Analysis of join-the-shortest-queue routing for web server farms. Performance Evaluation, 64(9-12), 1062–1081.* doi:10.1016/j.peva.2007.06.012

[9] Lu, Y., Xie, Q., Kliot, G., Geller, A., Larus, J. R., & Greenberg, A. (2011). *Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable web services. Performance Evaluation, 68(11), 1056–1071.* doi:10.1016/j.peva.2011.07.015

[10] Mukherjee, Debankur & Borst, Sem & Leeuwaarden, Johan & Whiting, Philip. (2016). Universality of Power-of-d Load Balancing in Many-Server Systems. arXiv. 8. 10.1287/stsy.2018.0016

[11] Vignesh Joshi. Load Balancing Algorithms in Cloud Computing. International Journal of Research in Engineering and Innovation, 2019, 3, pp.530 - 532. ffhal-02884073