© 2018 IJSRSET | Volume 4 | Issue 9 | Print ISSN: 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Fault Aware Load Balancing Algorithm

[1]S. Sivakumar, [2]V. Anuratha
[1]Research Scholar, Sree Saraswathi Thyagaraja College, Pollachi, Tamil Nadu, India
[2]Assistant Professor & Head, PG Department of Computer Science, Sree Saraswathi Thyagaraja College, Pollachi, Tamil Nadu, India

ABSTRACT

With the rapid growth in technology, there is a huge proliferation of data requests in cyberspace. Distributed system/servers play a crucial role in the management of request in cloud which is distributed among the various geographical zones. Many of time the system gets over loaded due to few of servers with high number of request and some of servers being idle. This leads to degradation of performance of over loaded servers and failure of requests. On these over loaded servers average response time of server increases. So there is a requirement to design a load balancing algorithm to optimize resource utilization, response time and avoid overload on any single resource. The management of data in cloud storage requires a special type of file system known as Distributed File System (DFS), which had functionality of conventional file systems as well as provide degrees of transparency to the user, and the system such as access transparency, location transparency, failure transparency, heterogeneity, and replication transparency.

Keywords : Distributed File System, Fault and Load Based Load Balancing Algorithm, Storage Servers.

I. INTRODUCTION

With the fast development in innovation, there is a tremendous expansion of information asks for in the internet. Dispersed framework/servers assume a pivotal part in the administration of demand in cloud which is appropriated among the different topographical zones. A considerable lot of time the framework gets over stacked because of few of servers with high number of demand and some of servers being inert. This prompts corruption of execution of over stacked servers and disappointment of solicitations. On these over stacked server's normal reaction time of server increments. So there is a necessity to outline a heap adjusting calculation to improve asset usage, reaction time and keep away from over-burden on any single asset.

The administration of information in distributed storage requires an extraordinary sort of document framework known as Distributed File System (DFS), which had usefulness of traditional record frameworks and additionally give degrees of straightforwardness to the client, and the framework, for example, get to straightforwardness, area straightforwardness, disappointment straightforwardness, heterogeneity, and replication straightforwardness. DFS gives the virtual deliberation to all customers that every one of the information found nearest to him. By and large, DFS comprises of ace slave engineering in which ace server keeps up the worldwide registry and all metadata data of all the slave servers. Though, slave speaks to a capacity server that stores the information associated with ace server and other stockpiling servers also. This stockpiling server handles the great many customers ask for simultaneously, in DFS. The heap appropriation of solicitations on these capacity servers is uneven and prompt execution debasement generally. Assets are not abused sufficiently, in light

of the fact that some server gets an excessive number of solicitations and some stay sit out of gear. In a dispersed stockpiling framework, load can be either regarding demands took care of by a server or capacity limit of that server or both. The fundamental commitment of this work is to enhance the normal asset use of framework and evacuating problem areas and chilly spots in the framework that is the unbalancing of solicitations over the framework ought to be evacuated.

II. FAULT AND LOAD AWARE LOAD BALANCING IN CLOUD STORAGE

In this approach, we have proposed a Fault and Load based Load adjusting calculation (FLA) that can adjust a heap of servers powerfully by thinking about its parallel preparing ability, handling time and its demand lining limit. Proposed calculation expects to enhance the execution distributed storage framework by lessening demand disappointment check, Average line length, normal use, and aggregate execution time.

2.1 Problem Statement

Distributed file system framework gives a typical virtual record framework interface to all clients as in DFS stockpiling servers are appropriated geologically and due to this heap dispersion of customer's solicitations to these servers end up uneven. This issue can be delineated plainly through Figure-1. Here, we have taken five stockpiling servers S_1, S_2, S_3, S_4 and S_5 with their separate administration rate (S_r) introduce in the framework. Administration rate of a server connotes the quantity of solicitations prepared by a server in a given time. At first at time t=0, we expect that every server gets an around measure up to measure of solicitations as appeared in Figure-1(a). We have taken aggregate 8 solicitations to delineate the situation of our concern explanation. In the second case as appeared in Figure-1(b) after time t=2, every server procedure the customer's solicitations according to its administration rate and server S_1 asks for gets over substantially sooner than different servers and S1 ends up sit out of gear. Server S_3 and S_5 are completely stacked and sets aside their opportunity to process all solicitations. From this situation, we can state that dispersed record framework does not use every server proficiently. In certifiable circumstance, these solicitations are too expansive as contrast with server benefit rate. So keeping in mind the end goal to build the framework execution a few solicitations which are in line should be moved to the sit out of gear servers or minimum stacked server and finishes the demand without disappointment. Our point is to keep away from line like circumstances, using the ability of every server productively and satisfy greatest demand without disappointment.

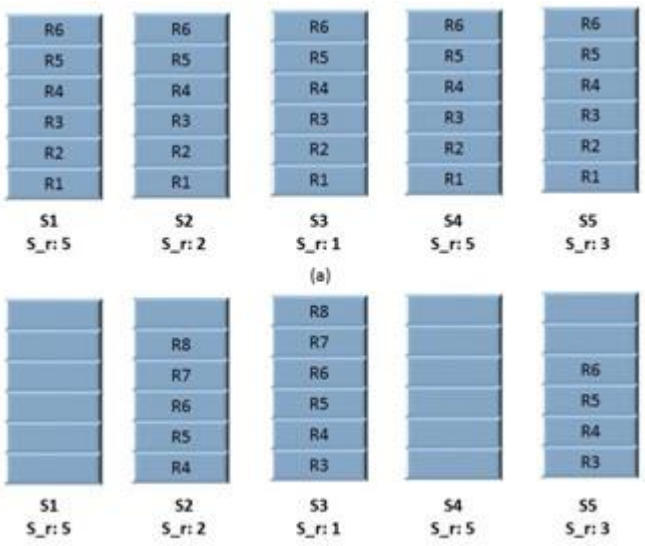

Figure-1 : Problem statement for load balancing (a) at time t=0, servers receive equal amount of client requests. (b) at time t=2, scenario of servers after processing the receive requests.

2.2 Problem Approach

Here, we have proposed a Fault and Load based Load adjusting calculation (FLA) that can adjust the heap of servers progressively by thinking about its parallel preparing ability, handling time and its demand lining limit. Proposed approach takes four fundamental parameters of a server 1) Server request queue size - buffer space to store the client requests to be handled

by the server. 2) Server service rate (λ) - the number of CPUs available for processing the client request in a server. 3) Processing time (S_T) – time takes to process a request which differs from server to server. 4) Fault rate. Modern servers are equipped with many features like multiple CPUs, large storage, high I/O capability etc. We have picked the different CPUs include as a principle parameter for stack adjusting of our proposed approach.

Following are the few assumptions that we have considered for our proposed approach:

- It is assumed that all the servers belong to same organization which can be geographically apart from each other. So each server maintains the replica of every server data.
- It is also assumed that all servers are strongly connected with each other through high bandwidth medium.
- Each server maintains global view which contains the information of its neighbors through master server.

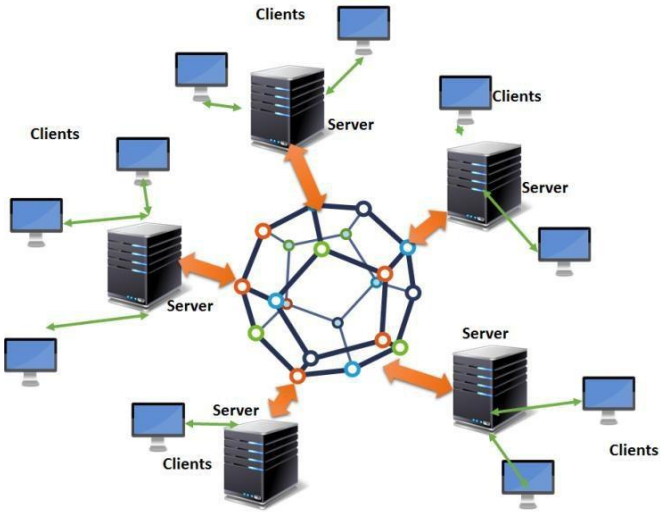

Figure-2: Organization of distribute storage servers.

Figure-2 demonstrates the general situation of circulated stockpiling servers. In Figure-2, there could be N associated servers where N {1,2,3 … .. n-1}, in the framework. Every server has following properties; for example, ask for line, number of CPUs, stockpiling limit. Customers send their solicitations to the particular server. Ordinarily the approaching solicitation rate (ρ) builds exponentially to a specific server. This is a result of the arrangement of customer's solicitations to that information that is put away inside the server. On the off chance that, when a server gets excessively numerous solicitations than server supports them in their demand line and the span of demand line gets increments powerfully just up to its predefined edge confine. Once, the demand line breaks as far as possible than server is considered as over-burden server and triggers the heap balancer. Load balancer orders the minimum stacked server based on their demand line and preparing limit. When the minimum stacked server gets grouped than over-burden server relocate its heap to that server and equalizations the heap. Various notations are used in the proposed approach and represented as follows:

ρ - Current queue size of server.

λ_i - Service rate that is number of request processed simultaneously on a server.

S_T - Service time is the time taken by server to process the request.

Q_L Current - Current queue length of server.

Q_L Threshold - Threshold limit of server request queue.

ΔL_i - additional load on server i.

W_i - Waiting time for a request at server i.

FT_i - Count of request failed.

FR_i - Fault rate that is the number of request failed due to system failure over time t.

F_j - Fitness value of neighbors of server i. (j ε {1,2,3 ….. n-1})

We have considered the real world scenario where the server request queue size and service rate changes with respect to time t dynamically and represented as δ_ρ and δ_λ respectively.

$$\delta_\rho = \frac{\rho}{\delta_t}$$

$$\delta_\lambda = \frac{\lambda}{\delta_t} \qquad (2.1)$$

Fault rate of a server can be given as:

$$FR_i = \frac{FT_i}{time} \quad (2.2)$$

Storage server is said to be overloaded if:

$$\delta_\rho > Q\_L_{threshold} \quad (2.3)$$

When server *i* where $i \in \{1,2,3 \ldots\ldots n-1\}$ is overloaded then it calculates the amount of extra load ΔL_i on that server which can be calculated as follow:

$$\Delta L_i = Q\_L_{current} - Q\_L_{threshold} \quad (2.4)$$

The condition when a load balancer module gets triggered on the overloaded server *i* is given below:

$$T(i) = \begin{cases} 1, & \Delta L > 0 \\ 0, & \text{otherwise} \end{cases} \quad (2.5)$$

T(i) is a triggering function.

Once, the load balancer module is triggered, server i find the least loaded or idle server that can accommodate its load and adequately process the service requests without failure. For this load balancer calculates the fitness value F_j that can be calculated using the following fitness function:

$$\Delta M_j = Q\_L_{threshold} - Q\_L_{current} \quad (2.6)$$

Here, ΔM_j is free request queue of server j. If ΔM_j is negative, then server j request queue is overloaded otherwise it is least loaded.

$$F_j = \alpha_1.\Delta M_j + \alpha_2.\lambda_j + \alpha_3.\left(\frac{1}{FR_j}\right) + \alpha_4.\left(\frac{1}{W_j}\right) \quad (2.7)$$

Here, α_1 and α_2 are constants and may vary according to scenario such that

$$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1 \quad (2.8)$$

For our proposed scenario, we have considered the value of α_1 and α_2 is 0.5 it is because both the parameters play the equal role in load balancing. In this way, load balancer calculates the fitness value for each neighbors of server *i.* and select that server which has maximum fitness F_j value, i.e. fault rate of server less than migrating server and migrate the ΔM_j amount of load to serverj. Selecting the server with maximum fitness value in turndecreases the failure probability of request and completes the request as soon as possible with least waiting time.

III. FAULT AWARE LOAD BALANCING ALGORITHM (FLA)

FLA calculations have been intended to adjust the customer asks for over the servers and appropriate the heap over the framework consistently. Here, stack balancer as appeared in Figure-3(a) routinely exams for the demand line measurement of server and attempts to limit the issue of over-burdening of any server with the guide of relocating the additional demand to other sit out of gear or slightest stacked and minimum flawed neighboring server in cloud. Proposed stack adjusting calculation is isolated into two phases. In first stage rundown of sit without moving servers is made, and in second stage the server with most elevated wellness esteem and which can satisfy the demand with slightest disappointment likelihood. The calculation checks and ascertains the wellness esteem for the neighbor server to store them in a rundown appeared in Figure-3(b). Load balancer uses this rundown to choose the server that has most astounding wellness esteem. Load balancer calculates the waiting time over each server from above list which can be given as:

$$W_k = \frac{Q\_L_{current\_k}}{\lambda_k} \text{ x } S\_T_k \quad (3.1)$$

This equation shows the W_k waiting time of i^{th} request at server 'k'.

In second stage load balancer then finds the server with least waiting time, least fault rate and highest service rate i.e. highest fitness value from the list. The proposed algorithm also tries to improve the server

response time by selecting the server having least CPU utilization. In this way, proposed algorithm utilizes the idle or underutilized server to increase the overall performance of the system and reduce requests failure over the system by reducing the probability of request failure.

Figure-3(a): FLA Load Balancing algorithm

Step-1: FLA(Server s, $Q\_L_{current}$, λ_k, S_T_k,FR_i)
Input: Server s, Queue length $Q\_L_{current}$,service rate λ_k,
service time S_T_k, fault rate FR_i
Step-2: s ← server
Step-3: $Q\_L_{current}$ ← current queue size
Step-4: λ_k ← service rate of server k
Step-5: S_T_k ← service time of server k
Step-6: FR_i ← fault rate of server k
Step-7: compute W_k
Step-8: **if** $(\delta_\rho > Q\_L_{threshold})$ **then**
Check server queue status;
Add request to queue;
Process_request();
else
Server is overloaded;
s ← Find server(server _ neighbour _ list L)
Find under loaded server;
s ← migrate request
Step-9: **Goto Step-7**
Output: Load balances the request

Figure-3(b): Find a neighbor server algorithm.

Step-1: Find_server(server_neighbour_list L)
Input: server_neighbour_list L
Step-2: For k:=1 to L.size()
Step-3: $s_1 \leftarrow L.get()$
Step-4: $F_k \leftarrow \alpha_1 \Delta M_j + \alpha_2 \lambda + \alpha_3 \left(\frac{1}{FR_j}\right) + \alpha_4 \left(\frac{1}{W_j}\right)$
Step-5: temp _ list $t \leftarrow F_k$
Step-6: **End For**
Step-7: L_2 = Sort(t);
Step-8: $s_2 \leftarrow \min(L_2)$
Step-9: return s_2

Output: The server with minimum fitness value.

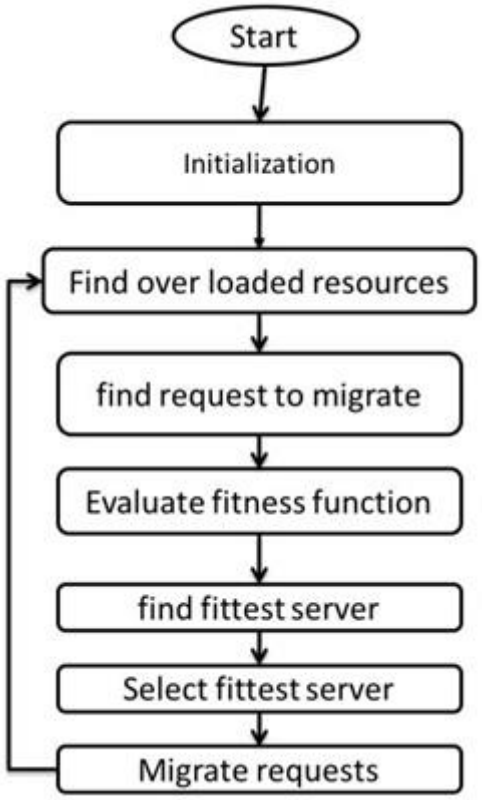

Figure-4 shows the flow of the algorithm with various phases of algorithm and interaction among them to find the fittest server for each request.

IV. RESULTS AND DISCUSSION

Execution examination of proposed FLA calculation is done utilizing CloudSim test system where we presently have a huge number of solicitations to be finished by 12 stockpiling servers. The majority of the servers work simultaneously with consistent amount of CPU centers to process the customer asks for quickly. Every server has a demand line to cushion the approaching customer demands, stockpiling capacity to store the information and satisfy the customer demands. For the given issue articulation, where the heap is uneven, it is accepted that half of capacity servers get customer solicitations and others stay sit. Our thought process is to similarly

appropriate the gotten customer demands among the servers to maintain a strategic distance from the situation of over-burdening. In the reenactment situation quantities of capacity servers are kept settled with changing number of solicitations taking care of. We have additionally contrasted the gotten comes about and the slightest load adjusting calculation. Following table delineates the design parameter for our recreation condition.

No. of Client Request	No. of Servers	No. of CPU cores available per server	Storage Capacity of server in GB	Server Queue length
800	12	7	500	20
1000	12	7	500	20
1200	12	7	500	25
1800	12	7	500	25
2400	12	7	500	25

Table 3.1: Experimental parameters used for simulation environment

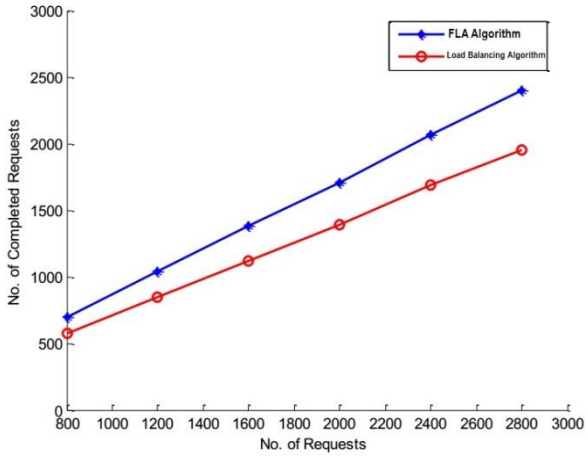

Figure 3.5: Number of request completed

Figure 3.5 shows the number of processed client requests by server in a given time. Here, Figure 3.5 represents the graph between numbers of sent requests vs. numbers of completed request whereas Figure 3.6 represents the graph between no. of sent requests vs. no. of failed requests for the proposed and least load algorithms. In least loaded algorithm when any server get overloaded then load balancer selects the server of which request queue is least loaded without considering the CPU parameter.

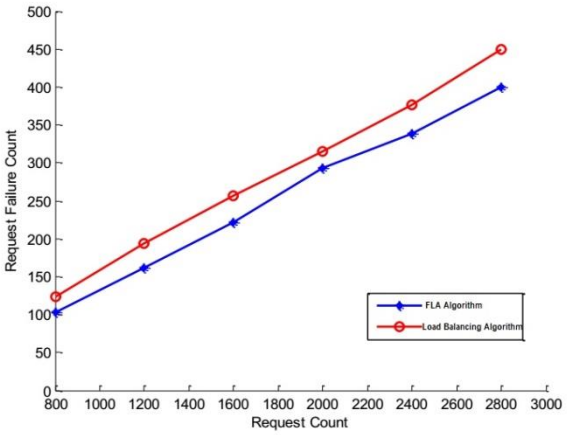

Figure 3.6: Number of sent requests vs. no. of failed requests.

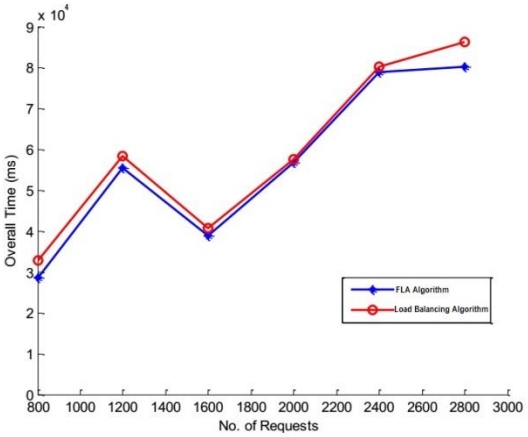

Figure 3.7: Overall response time.

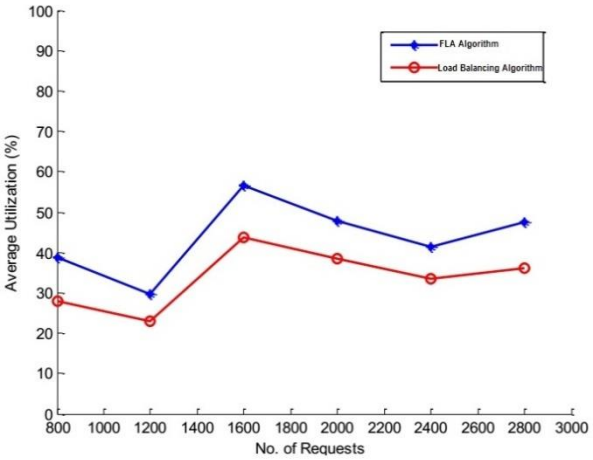

Figure 3.8 Average utilization of system.

For the proposed algorithm we have considered the CPU parameter and from obtained results as shown in Figure 3.5, Figure 3.6 and Figure 3.7 that the proposed algorithm perform much better over the least load algorithm. Figure 3.8 shows that the proposed FLA algorithm improves the average utilization of the system drastically over increasing requests due to improvement in total request completed. In all set of client requests, proposed algorithm process more number of client's request with better overall response time as shown in Figure 3.7.

V. CONCLUSION

The main achievement of this work is to find the rich literature and solve the issue of load balancing in fault aware cloud environment. In distributed file system, data is dispersed among different storage servers located geographically far away from each other. To provide the desired quality of service to the clients, performance of the distributed file system matters a lot. Response time is the major parameter that may affect the performance of the any distributed file system. Proposed approach claims to reduce the delayed requests and also reduces the overall system response time. The first approach also considers the physical aspects of a server like available number of CPU cores in a server, request queue size or buffer to store the incoming client requests. Moreover the second approach also considers the deadline of client requests to reduce request failure due to deadline. Obtained result shows the improvements over previously worked least loaded algorithm and more number of client requests are processed by the system without delay and in case of overloading and failure the load balance distribute the requests accordingly to neighbor servers. The results obtained from our approaches are very competitive with most of the well known algorithms and justified over the large collection of requests. Proposed load balancing algorithm proves to provide better fault tolerance as compared to existing algorithm with least request failure, reduced average utilization, average delay and high request completion count.

VI. REFERENCES

1. Abdul rahman and A. Almutairi, "A Cloud Access Control Architecture for Cloud Computing", 9(2), 2012.
2. ACM. Fong, Baoyao Zhou, Hui S.C., Hong G.Y. and The Anh Do, "Web Content Recommender System based on Consumer Behavior Modeling", IEEE Transactions on Consumer Electronics, 57(2), 2011.
3. Anuj Sehgal, "Introduction to OpenStack", 6th International Conference on Autonomous Infrastructure, Management and Security, 2012.
4. Arshad J, Townend P and Jie Xu, "Quantification of Security for compute Intensive Workloads in Clouds", 15th International Conference on Parallel and Cloud Systems, School of Computation, Pp. 478-486, 2009.
5. Artur A, Kondo D and Anderson D P, "Exploiting Non-Dedicated Resources for Cloud Computing", IEEE Network Operations and Management Symposium, PP. 341-348, 2012.
6. At&T Cloud Architect, Downloaded from http://cloudarchitect.att.com/Home.
7. Awang, N.F.B, "Trusted Computing - Opportunities & Risks", 5th International Conference on Collaborative Computing: Networking, Applications and Work sharing, Pp.1-5, 2009.
8. Barsoum, A. and HASAN A, "Enabling Dynamic Data and Indirect Mutual Trust for Cloud Computing Storage Systems",IEEE Transactions on Parallel and Cloud Systems, 2012.
9. J Basu and V Callaghan, "Towards A trust based Approach to Security and User Confidence in pervasive computing Systems", The IEE International Workshop on Intelligent Environments, Pp. 223 - 229, 2005.
10. Azzedine Boukerche, Yonglin Ren, "A trust-based security system for ubiquitous and pervasive computing environments", Commputer communications, 2008.