DOI: http://dx.doi.org/10.26483/ijarcs.v8i7.4371

ISSN No. 0976-5697

Volume 8, No. 7, July – August 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

PERFORMANCE EVALUATION OF DYNAMIC LOAD BALANCING SYSTEM BY USING NUMBER OF EFFECTIVE PARAMETERS

Deepti Sharma & Vijay B. Aggarwal
Dept. of Information Technology
Jagan Institute of Management Studies, GGSIPU, Rohini, Delhi, India

Abstract: Today, the World Wide Web is growing at an increasing rate and occupied a big percentage of the traffic in the Internet. These systems lead to overloaded and congested proxy servers. Load Balancing and Clustering of web servers helps in balancing this load among various web servers. In this paper we have given solution for load balancing among web servers and evaluated their performance using number of effective parameters. This paper explains the main objectives of proposed algorithm. It shows architecture diagram for load balancing in web server clusters. Proposed algorithm is explained using Pseudo code. Time complexity of proposed algorithm is also shown. Finally, it is concluded with experimental results andanalysis.

Keywords: DynamicLoad Balancing; Cluster System; load sharing and web traces.

1. INTRODUCTION

Accessing web sites today has many challenges in terms of response time, throughput and resource utilization. There is a heavy load on web servers and this load has to be distributed among these web servers[1]. Load Balancing among web servers is one of the solutions for solving these problems. There are several methods that have been proposed in the literature related to load balancing in web server clusters [2] [3].These approaches have already been discussed in past and the discussion provides motivation for us to continue the work in this field. In this research paper, we present our research work consisting of a new algorithm, viz. Performance Evaluation of Dynamic Load Balancing System by Using Number of Effective Parameters. The proposed algorithms produce solutions to the problems which have been faced by other basic algorithms. In this paper, we discuss our algorithm along with its objectives, architecture, time complexities and experimental results.

2. PROPOSED METHOD

In this approach, we propose a new request distribution algorithm for load balancing among web server clusters. The Dynamic Load Balancing among web servers take place based on user's request and dynamically estimating server workload using multiple parameters like processing and memory requirement [5] [6]. Our simulation results show that, the proposed method dynamically and efficiently balance the load to scale up the services, calculate average response time, average waiting time and server's throughput on different web servers[11]. At the end of this paper, we presented an experimentation of running proposed system which proves the proposed algorithm is efficient in terms of speed of processing, response time, server utilization and cost efficiency.

This framework uses the dynamic algorithm to analyze the current system load and various cost factors in arriving at the best target processor to handle the job processing. This algorithm uses a load factor to distribute the jobs among the web servers. Load factor is decided on the basis of processing capabilities of servers. The server with highest configuration will be assigned as highest load factor; lowest configuration server will take less load factor and so on. This can be implemented on proposed framework. Whenever there is a request from client, it will be distributed among web servers based on load factor.

The prime contribution of this research is to propose a framework that can run web server cluster system based on gathered requirements and to present its implementation on one of the web server in the cluster system. Another contribution is to present the architecture diagram and its implementation followed by an experimental analysis to prove the proposed research based on the various parameters such as speed of processing, response time, server utilization and cost efficiency [11].

3. ARCHITECTURE DIAGRAM FOR LOAD BALANCING IN WEB SERVER CLUSTERS

Web server allows running an application on several servers in parallel [4]. The load is distributed among different servers in a cluster. The application is accessible on other nodes of cluster even if any server fails. These clustering solutions provide scalability, high availability and load balancing [7]. The main objective of load balancing is to provide the best possible response time to user by distributing load using the servers in the cluster [9]. In this type of solution, the architecture involves use of load distribution algorithm, like simple round robin algorithm or other refined algorithms [8]. These algorithms help to distribute requests to other servers in the cluster taking load status and available resources on the servers.

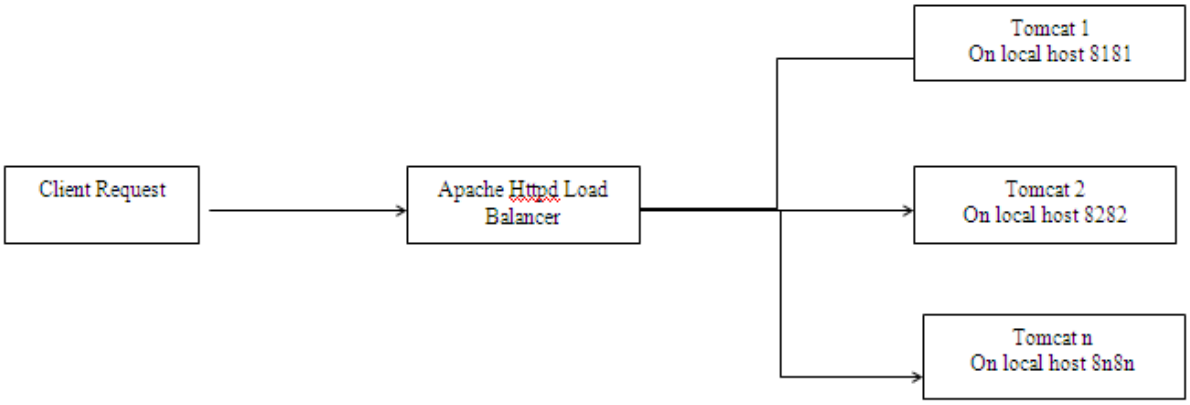

Figure1. Architecture Diagram for Load Balancing among 'n' Web Server Cluster and one Apache Load Balancer

A. Details of Architecture Model

In the above architecture, the request will come from the browser. There may be 'n' instances of tomcat web servers depending upon the requirement. In utmost cases the java web applications will run on different Tomcat instances and multiple server machines. But for simulation in our proposed approach, they are placed on a single machine. Single web application is deployed in all tomcat instances. There is one Apache httpd web server that works as load balancer with mod_jk module [10]. It is placed in-front of all instances of tomcat web servers to accept the request and distribute to these back end web servers based on load factor. These tomcat web servers are managed by the same apache load balancer. When request comes from client, it will be assigned to load balancer which will be checked further and be load-balanced based on their load factors.

4. PROPOSED ALGORITHM – Pseudo code

As discussed above, in this proposed algorithm, load is distributed among 'n' web servers on the basis of load factor. Following Pseudocode shows the working of algorithm. It first initializes the server and load balancer. When request will come from clients, it will go to load balancer first which is placed in between client and other tomcat servers. Load balancer module will check and distribute this dynamic load among web servers based on load factor.

Begin:

1. **Server Initialization**

a. Three tomcat servers are initialized based on load factor (LF)
b. Load Factor is assigned as LF=4 to WS1, LF=2 to WS2 and LF=1 to WS3

2. **Load Balancer Initialization**
3. **client_module ( )**

```
{
callclient.request ( ) /* forwards the client request
*/
}
```

4. **loadbalancer_module ( )**

```
{/*Distribute requests as per load factor (LF =4 or 2 or 1)

WS 1 will get maximum jobs,
WS 2 will get medium jobs and WS 3 will get least jobs.*

for (int j = 0 ; j <noOfLoops ; j++) {
ExecutorService executor =
Executors.newFThPool(noClients);
//creating a pool of 5 threads

for (int i = 0; i <noOfClients; i++) {

Runnable worker = new Client( );
executor.execute(worker); //calling execute method of

ExecutorService
} } }
```

5. **Call jk_status manager**
    a. Displays job distribution among 'n' tomcat web servers
    b. Shows errors, number of failed requests and client errors (if any)
    c. Displays current and maximum number of busy connections
    d. Shows number of bytes read and written

End

5. EXPERIMENTAL RESULTS AND ANALYSIS

This section depicts the experimental results obtained from the proposed approaches as well as motivating approaches. The Proposed approach has been implemented in Java. It has been validated on real time requests (ref: http://cricscore-api.appspot.com/) and two load balancers namely Apache and Nginx. The model is validated by using different number of requests ranging from 100 to 10000. For the purpose of validation, different parameters are defined:

- Availability: Percentage of time that the server has been up
- Health Check: Most recent and most severe logs, errors, etc.
- Activity: Throughput over the last minute including requests, bytes, etc.
- Resource Utilization: Resources currently in use and still available. It shows current busy connections and maximum busy connections.
- Efficiency: Server optimizations to maximize throughput. Sticky sessions.

The results are tested and run on two load balancers namely Apache and Nginx. The difference between two are given in following table.
The obtained results are given in the following sub-sections.

A. Results with Apache and Nginx Proxy Load Balancer

The above approach is tested and run on two load balancers namely Apache and Nginx. The total processing time taken (in sec) has been measured and compared. The effect on total processing time with change in number of requests can be checked from the table 1 and figure.

Table I. Total Processing Time with change in Number of Requests

No. of Requests	***Total Processing Time (sec)***	
	Apache Load Balancer	***Nginx Load Balancer***
100	14.7	8.39
400	22.8	18.05
900	52.2	36.6
1600	177.2	59.5
2500	192.69	124.31
6400	292.52	186.78
10000	538.55	346.09

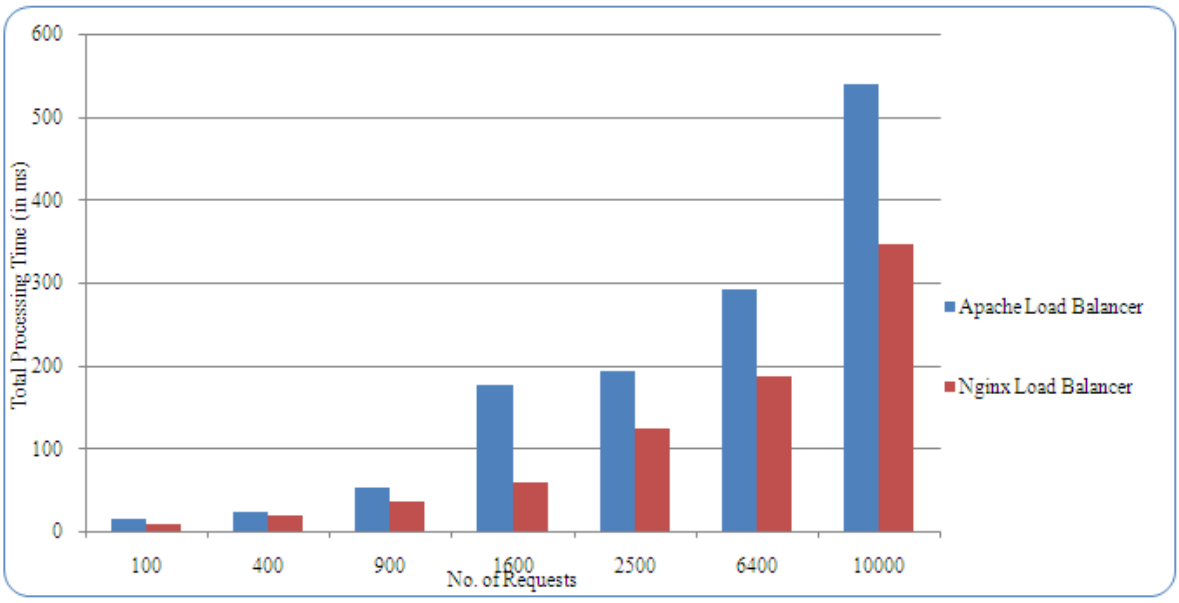

Figure2.Graphical Representation of above table.

Above table and Figure illustrates the total number of requests made from the users and its corresponding total processing time on Apache and Nginx Load Balancer. It shows that the total processing time is less with Nginx as compared to Apache.

6. TIME COMPLEXITY OF PROPOSED ALGORITHM

Time complexity of an algorithm is defined as the amount of time taken by any algorithm to complete all instructions or statements. It is expressed using big O notation. In our proposed approach, we concentrate on finding complexity of load balancer module which actually distributes the load among web servers.

Time Complexity for Algorithm 1

```
loadbalancer_module ( )
  {
  for ( j = 0 ; j <noOfLoops ; j++ ) {// O(n) – where ' n' is
  no of threads
  ExecutorService executor =
  Executors.newFixedThreadPool(noOfClients);  // O (1)
  for (i = 0; i <noOfClients ;  i++) {  // O(m) – where ' m'
  is no of requests
  Runnable worker = new Client( );  // O (1)
  executor.execute(worker);           // O (1)
        }
  }}
```

So overall complexity of Algorithm is O (n m)

7. REFERENCES

[1] H. Anna, J. Theodore, “A Study of Dynamic Load Balancing in a Distributed System”, SIGCOMM '86, Proceedings of the ACM SIGCOMM conference on Communications architectures & protocols, 1986.

[2] Md. J. Zaki et al, “Customized Dynamic Load Balancing for a Network of Workstations”, Journal of Parallel And Distributed Computing 43, 156–162 (1996).

[3] J. P. Andrew and J. Thomas Naughton, “Framework for task scheduling in heterogeneous distributed computing using genetic algorithms”, Artificial Intelligence Review, Volume 24, Issue 3, pp 415-429, 2005.

[4] Z. Lan, V. E. Taylor and G. Bryan, “Dynamic Load Balancing of SAMR Applications on Distributed Systems”, Supercomputing, ACM/IEEE Conference, 2001.

[5] S.Sharma and J.Godara. "Load Balancing in Cloud Computing”, International Journal of Computer Systems (IJCS), pp: 322-326, Volume 3, Issue 4, April 2016.

[6] L. Nguyen, and J. L. Larson. "Providing on-demand capabilities using virtual machines and clustering processes", U.S. Patent No. 7,577,959. 18 Aug. 2009.

[7] S. K. Stephen et al, "A decision framework for cloud computing", System Science (HICSS), 45th Hawaii International Conference, IEEE, 2012.

[8] A. A. Rajguru, and S. S. Apte. "A comparative performance analysis of load balancing algorithms in distributed system using qualitative parameters", International Journal of Recent Technology and Engineering (IJRTE), ISSN 1.3: 2277-3878, 2012.

[9] P. K. Singh, “An Efficient Load Balancing Algorithm in Distributed Network”, Diss. Jaypee University Of Engineering & Technology, 2015.

[10] http://hosteddocs.ittoolbox.com/AP121907.pdf

[11] Deepti Sharma, Vijay B. Aggarwal, “Improving Performance of Dynamic Load Balancing among Web Servers by Using Number of Effective Parameters”, published in International Journal Information Technology and Computer Science (IJITCS), 2016, 12, 27-38, Published Online December 2016, DOI10.5815/ijitcs.2016.12.04. http://www.mecs-press.org/ijitcs/ijitcs-v8-n12/IJITCS-V8-N12-4.pdf