An Effective Mechanism for Improving Performance of Load Balancing System in Cluster Computing

Deepti Sharma,
Asst. Professor (IT)
Department of Information Technology,
Jagan Institute of Management Studies
Affiliated to GGSIPU, Rohini, Delhi

Vijay B. Aggarwal, Ph.D.
Professor (IT)
Department of Information Technology,
Jagan Institute of Management Studies
Affiliated to GGSIPU, Rohini, Delhi

ABSTRACT
With the fast growth of the WWW, the traffic to the websites is also increased. Due to which, clients many times experience poor response time or sometimes denial of service. These bottlenecks of the server can be solved using multiple web servers that behave like a single host. With the rapid growth of both information and users, the quality of network services should be improved. Server's performance can be enhanced using load balancing mechanism which is the process of redistributing the work load among processors in the system. To evenly distribute the load among the web servers, dynamic load balancing (DLB) techniques are used. In this paper, a DLB algorithm is given which will calculate, allocate and balance the load among web servers on the basis of processing capacity and memory requirement of jobs. Performance of the proposed framework is also analyzed.

Keywords
Web Servers, Load Balancing, Response Time, Throughput

1. INTRODUCTION
In today's world, we have habit of living with gadgets around us. They are needed for the mobile recharges or bill payment, ordering a pizza, seeing a video on line, and many more. There is so much dependency on the gadgets that all social networking sites are used for making virtual friends or sending messages through sites. For all these, Internet is required. Now a day, Internet is the necessity as well as requirement. An Internet enabled device has the capacity to interact with world virtually and gets the task done easily and quickly. On Internet, various heterogeneous devices are required with different configuration, different operating systems, and different bandwidth. With this, Internet is becoming more and more complex, both in terms of, size as well as traffic over the Internet. To access Internet, request is generated through client's browser and it is served by web server or a cluster of web servers. A Cluster of web servers can refer to either the hardware (the computer) or the software (the computer application) that helps to deliver web content that can be accessed through the Internet [1].

Today, people who use Internet want the response from web site as fast as possible. If the response time of a web site is too slow or not able to process the request within the time span, it is assumed that this web site is too slow, and surfing that web site again is avoided. A client always prefers the sites which has quick response time. Thus there is need for a distributed processing system so that smaller and inexpensive heterogeneous computer systems should be utilized to achieve the required computation. Such systems are usually independent with their own memory and storage resources, but connected to a network so that the systems communicate with each other for sharing the load. In such systems, the major task of a centralized monitor is to distribute jobs among web servers [2]. Due to heterogeneous machines, the central monitor usually keeps track of the load on each such system and assigns tasks to them. To manage the overloaded requests, a facility is required that will ensure that the requests assigns to the web server is properly handled. Further, if more number of requests comes, they can be further distributed among different web servers [1][4]. Over a period of time, the performance of each system may be identified and the information can be used for effective load balancing. To distribute jobs to various nodes so as to derive maximum efficiency and minimum wait time for jobs, various factors are used to consider like network latency, I/O overhead, job arrival rate and processing rate[5].

Also all these web servers should be load-balanced. The main task of load balancing is to distribute the tasks to the web server on evenly basis. **Load Balancing can be defined as to balance the data's in and out from the server [2].** Load balancing are of two types; when constant workload is assigned for the computation to the processor, this is called *static load balancing*. And, when there is a variable workload for the computation and that can be changed during computation that is said to be *dynamic load balancing [2]*. In static load balancing work is distributed statically without giving emphasis on runtime events. This will leads to a stage where it is impossible to judge the work load at the initial stages for the future usage. But, in dynamic load balancing, every time the new workload arrives, the distribution of work load by the master processor is done dynamically. In static load balancing the performance is best. But in dynamic load balancing, Load Balancing can be used in the best way to process work load dynamically [6].

To evenly distribute the load among the servers under clusters, dynamic load balancing (DLB) techniques are used. DLB optimizes request distribution among servers based on factors like server capacity, current load level and historical performance [8]. It also improves mean response time and overall throughput of Web Server Clusters (WSC). To further improve the performance of the WSC, there are different scheduling techniques e.g. round robin (RR), weighted round robin (WRR), shortest queue (SQ) etc [2].

To improve the system performance and overall throughput, we have developed a framework in which load will be balanced based on memory and processing speed requirement. The framework and formal description of the algorithm is described in the following sections.

2. APPROACH
In this article, we will discuss about our approach for load balancing mechanism. In this, we are taking consideration that there are 'n' numbers of servers. The value for 'n' is variable.

Server is having basic parameters as server's memory, processing speed and memory left over. Initially, server's memory and memory left over are same. Server's memory leftover will fluctuate if there is any job allocation to server or completion of job is done. Total numbers of jobs are designated by 'm'. Jobs generated for an individual interval can be defined as:

J[i]= k, where J = { J_1, J_2…J_n}, 'i' is the interval generated and k is the job generated for the respective interval and k={0, 1….x} where 'x' can be defined as maximum number of jobs that can be generated for an individual interval. Job parameters are job memory, job processing speed and total expected execution time. Here, job's memory and processing speed means how much memory and processing of the server required by job for execution. Job's expected execution time is the maximum time required by the job for execution.

In our approach, total time is divided into intervals and there is a fixed time slice of 5 milliseconds. At the time of initialization of intervals, jobs are generated, initialized and allocated to the servers. These jobs are recorded in "new job queue". In addition, there are three types of queues.

They are defined as follows:

a) *Running Queue*: contains all the running jobs.

b) *New Job Queue*: contains all the jobs that are generated, when the interval begins.

c) *Waiting Queue*: the jobs that are initialized, but waiting for allocation to server and execution.

There are various processes also running in the system. They are as follows:

i) ***Allocation Process*:** Jobs are allocated when the job's memory and processing requirements are satisfied by server's memory left over and processing speed. All servers would be checked for condition, if conditions are satisfied, the job is allocated to the respective server. If all the servers are checked and job is still unallocated, it will be in waiting queue.

ii) ***Load Balancing (LB)*:** In our approach, LB is taken care at the time of job allocation. While allocating job, server's memory left over and processing speed is checked. If it is greater than the job's memory and job's processing requirement, then only it is being allocated to respective server. This makes the respective server even, or load balanced.

iii) ***Job Completion Process*:** Remaining expected execution time (initially same as maximum expected execution) is decremented with the reduction value of the respected server (to which job is allocated) after every cycle. If it becomes zero or less than zero, job is completed and done. At the same time, the memory left of that server is incremented with the value of job that is currently completed.

iv) ***Reduction Process:*** Reduction process is a process used to calculate reduction value, associate with the respective server. The reduction value is the value with which job's expected execution time was decremented. It was differ from server to server, on the basis of the processing speed of respected server.

At last, when numbers of intervals are over, but still there are jobs left for execution. In this case, execution will be continued until there is no job left for the execution.

3. DESIGN

3.1 Diagram

Heterogeneity implies heterogeneous types of devices, operating systems etc. There may be heterogeneous devices requesting from client's machine to access web server. But, basic functionality to connect with the requested web site is same through Internet. Server Controller receives the request for the web site and forwards it to the web server. It distributes the requests in such a manner that all the web servers would be load balanced.

As shown in Figure 1, request is coming from various client machines and is balanced among various web servers through Server Controller. "DNS resolve process" is shown in Figure 2 below. There may be 'x' number of clients. At any point of time, they may be connected to any website. Once a request is generated from the client, it is broken into Domain Name and thereafter an IP is generated which is forwarded to the respective web site's server. The domain name system (DNS) is the way that Internet domain names are located and translated into Internet Protocol addresses. A domain name is a meaningful and easy-to-remember "handle" for an Internet address.

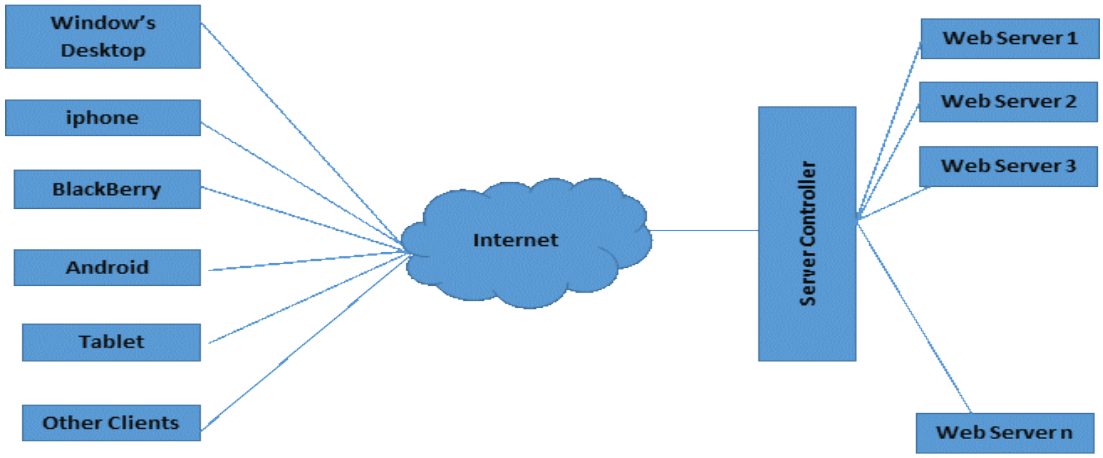


Figure 1: The model of web server system with N nodes

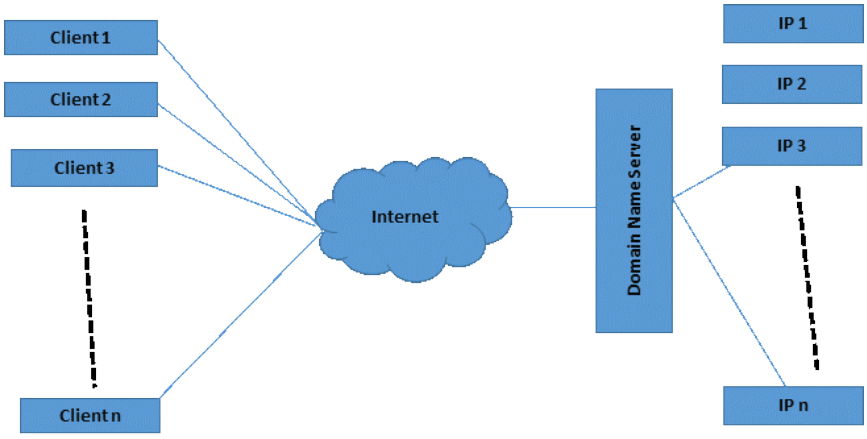


Figure 2: DNS Resolve Process

3.2 Flow Chart

Flow chart is a graphical representation of a computer program in relation to its sequence of functions (as distinct from the data it processes).

The flow graph given below depicts the overall steps of the proposed approach in pictorial form.

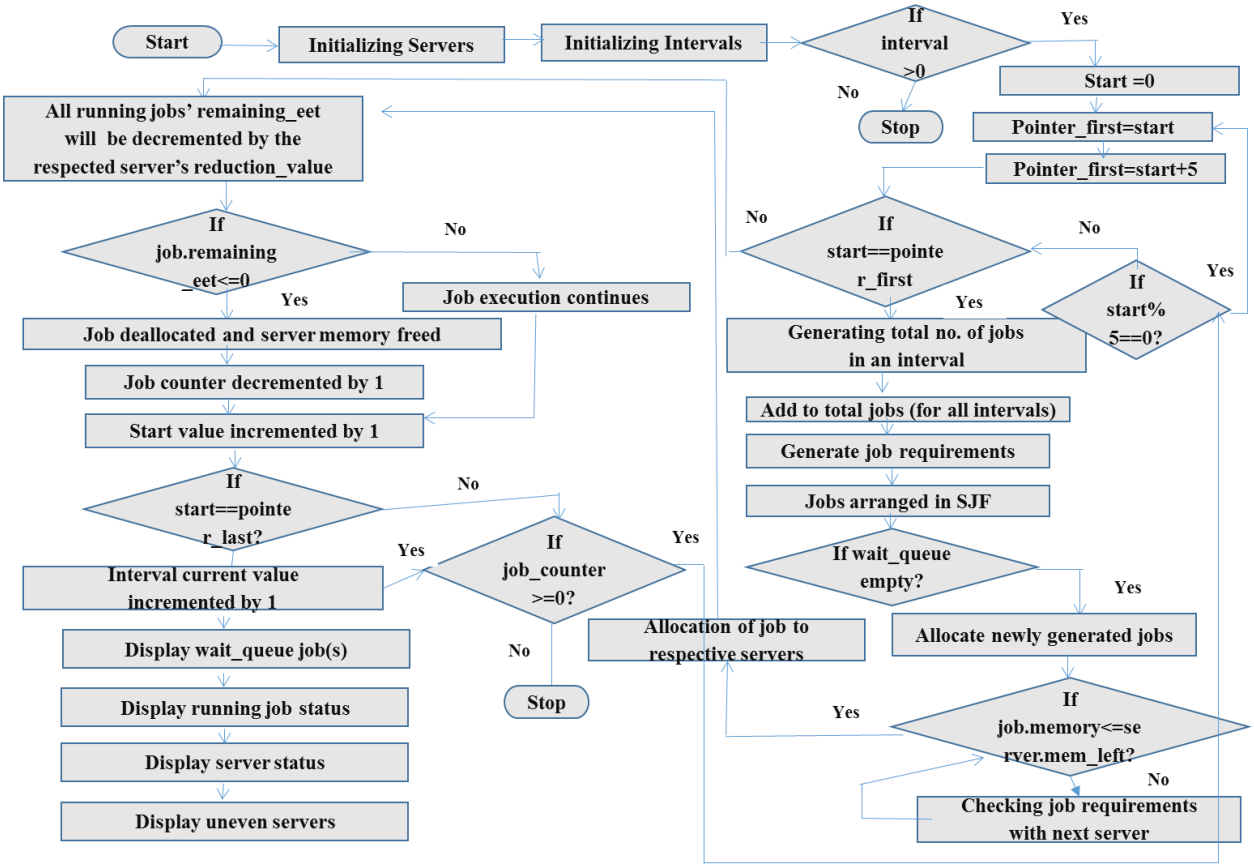


Figure 3: Flowchart shows overall representation of proposed approach

4. FORMAL DESCRIPTION OF THE ALGORITHM

Terms to be used in the algorithm are as follows:

1. *Time Interval*: The amount of time between two specified instants, events or states. Our algorithm uses the concept of time interval which would be generated randomly for every 5 milliseconds.

2. *Job*: A task performed by computer system. Jobs can be performed by a single program or by a collection of programs. In our algorithm, jobs are generated randomly in each time interval. Jobs that are generated have various parameters like

processing time, memory requirement and total expected execution time.

3. *Server*: A computer program which manages access to a centralized resource or service in a network. Servers have various parameters like processing speed, memory, memory left and jobs assigned.

Following are the tables generated for Intervals, Jobs, Servers and allocations of jobs.

Table 1: Intervals Generated

Intervals	Jobs Generated*
0	5
1	15
2	18
3	7
4	8

Table 2: Jobs Generated per Interval

Job_id	Job_memory	Job_processing	J_E_E_T*
1	870	41	4
2	1032	68	13
3	79	45	6
4	42	56	7
5	512	80	4

Table 3: Server's Detail

Server_Id	Server_Memory	Server_Processing	Jobs_Allocated
1	500	50	10,5,27,4,9,…..
2	1000	100	0,11,7, 18, 1,….
3	1500	150	3, 3, 30, 28, 22, ….
4	2000	200	6,2,22,14,6,5,….
5	2500	250	9,1,8,25,….

Table 4: Allocation & Load Balancing

Job_id	Server_Id
1	5
2	4
3	3
4	1
5	1

4.1 Informal Description of the Algorithm

1. Initializing the servers (with all parameters defined above)

2. Initializing Intervals (maximum number of intervals will be defined)

3. At the starting of interval: if start == pointer_first

a) Generating total number of jobs in each interval

b) Generating job requirement (with all parameters defined above) in each interval

c) Jobs would be arranged in SJF manner on the basis of total expected execution time.

d) Introduction of jobs in the waiting queue, if any, for allocation

e) Start allocating jobs on the basis of below condition:

f) if ((job.memory<=server.memory_left)AND(job.processing_requirement < = server.processing))

i. Allocate job to respective server

ii. Increment the Job Counter of the respective server

iii. Add job_id to the job array of the respective server.

else

i) add element to waiting_queue

ii) increment the value of wait_counter

iii) remove the element of array, my_job_new

iv) decrement the value of total jobs of an interval

4. Different job queues are maintained for:

a) Already running jobs

b) Newly created jobs

c) Waiting jobs

5. Performing Load Balancing (LB). LB would be working concurrently while allocating the jobs to the servers. The condition for LB is if (server.memory_left < 0). That is, if for any server the memory left over is less than zero, it will not be allocated to any server and thus servers are balanced and even.

6. When the jobs is completed and done, on the basis of their remaining expected execution time. Remaining expected execution time is calculated at every millisecond. Once it becomes zero, it will be completed from the server. Also, the job's memory would be added to server's memory_left and it may be allocated to other jobs waiting in queue.

7. At the end of interval,

If start= =pointer_last

a) interval_current value will be incremented by 1

b) displaying of:

i. the WAIT_QUEUE array

ii. Status of WAIT_QUEUE job/jobs.

iii. The JOB_ARRAY (currently running Job/Jobs).

iv. Status of JOB_ARRAY job/jobs.

v. Status of all servers (0 - Idle, 1 - Busy)

vi. Status of UNEVEN server/servers (if any).

8. Finally, display

a) The job status of currently running jobs

b) Status of all servers

c) Status of uneven servers (if any)

9. Repeat steps 5 to 10 until all jobs that are generated would be completed and done.

5. SIMULATION AND EXPERIMENTAL RESULTS

For simulating the design, a Java based processing system is implemented with multiple processors, jobs, time intervals and web servers. As part of this experiment, four cases are defined with maximum 20 intervals, 80 jobs/interval and 10 servers. In this, total number of jobs and intervals are generated randomly and the number of web servers is fixed. We have calculated Execution time w.r.t jobs and intervals generated. In the experiment, they are defined as:

T.I.G.: Total Interval Generated, T.J.G.: Total Job Generated, T.S.: Total Server, E.E.T.: Expected Execution Time and F.E.T: Final Execution time

5.1 Results with Different Intervals, Jobs and Servers

Experiments are performed on the algorithm in four cases, and they are defined in the table below:

Table 5: Resulted Table

Cases	T.I.G.	T.J.G.(Per Interval)	T.J.G(All Intervals)	T.S.	E.E.T	F.E.T.
1	20	0-50	546	10	0-20	109
2	20	0-80	954	10	0-20	136
3	20	0-50	463	8	0-20	123
4	20	0-50	473	10	0-40	125

Finally, the values derived were plotted as a graph shown in Figure 4.

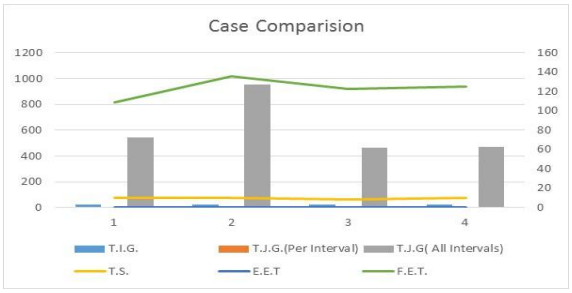


Figure 4: Graphical Representation of Resulted Table

5.2 Results with Jobs in Waiting Queue

Table 6 shows the jobs in waiting queue in each interval(0-24) and Figure 5 shows its graphical representation.

Table 6: Jobs in Waiting Queue

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Case 1	0	0	0	3	19	3	9	15	22	28	17	2	0	0	7	7	18	33	23	7	0	0	0	0	0
Case 2	35	54	51	55	52	91	70	91	130	55	83	117	142	160	195	117	171	207	230	197	177	97	84	44	5
Case 3	0	0	0	23	37	41	33	42	37	35	56	40	42	6	30	40	59	53	72	73	38	15	0	0	0
Case 4	0	0	8	11	0	0	11	9	3	0	0	0	16	29	23	35	35	19	40	46	6	0	0	0	0

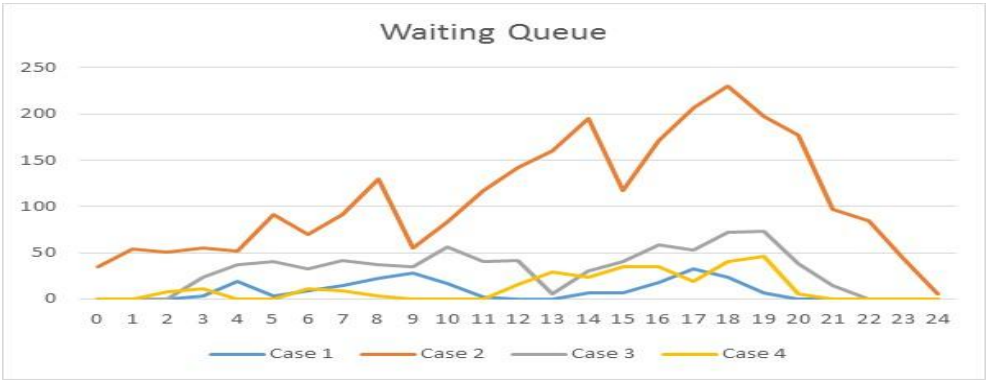


Figure 5: Graphical Representation of Table 6

5.3 Results with Server's Status per Interval

Following table shows the Server's Status in each interval.

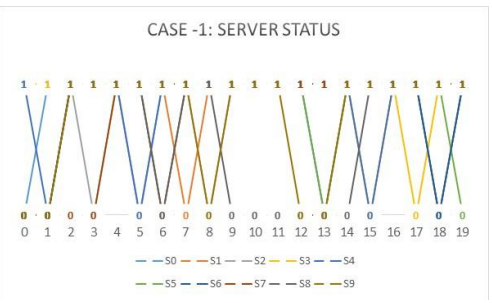
Table 7: Case 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
S0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
S1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0	1
S2	1	1	1	0	1	0	1	1	0	1	1	1	1	1	1	0	1	1	1	1
S3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1
S4	1	0	1	1	1	0	1	1	1	1	1	1	1	0	1	0	1	1	0	1
S5	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0
S6	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
S7	0	0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
S8	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	1
S9	0	0	1	1	1	1	1	1	0	1	1	1	0	0	1	1	1	1	1	1

Case -1: Server: 10 (S0-S9), Total Job Generated: 0-50 per interval, Interval: 20 (0-19), EET: 0 - 20

The data collected is listed in Table 7 (0 - Idle, 1 – Busy) and shown in Figure 6.

Figure 6: Graphical Representation of Case 1



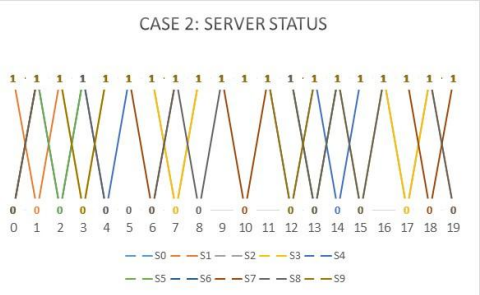
Case 2: Server: 10, Total Job Generated: 0-80 per Interval, Interval: 20, EET: 0-20

The data collected is listed in Table 8 (0 - Idle, 1 - Busy) and shown in Figure 7.

Table 8: Case 2

Interval	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
S0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
S1	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0
S2	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0	1	0	1	1
S3	0	1	1	0	1	1	1	0	1	1	1	1	1	1	1	0	1	0	1	1
S4	1	1	0	1	0	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1
S5	1	1	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
S6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
S7	0	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	0	1
S8	0	1	1	1	0	0	0	1	0	1	1	1	1	0	1	0	1	1	1	0
S9	1	1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1

Figure - 7: Graphical Representation of Case 2



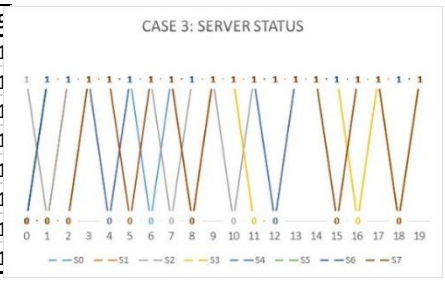
Case 3: Server: 8, Total Job Generated: 0-50 per Interval, Interval: 20, EET: 0-20

The data collected is listed in Table 9 (0 - Idle, 1 - Busy) and shown in Figure 8.

Table 9: Case 3

Interval	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
S0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
S1	0	0	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
S2	1	0	1	1	1	1	1	0	1	1	0	1	0	1	1	0	1	1	0	1
S3	0	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0	1	0	1
S4	0	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
S5	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
S6	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
S7	0	0	0	1	1	0	1	1	0	1	1	1	1	1	1	0	1	1	0	1

Figure - 8: Graphical Representation of Case 3



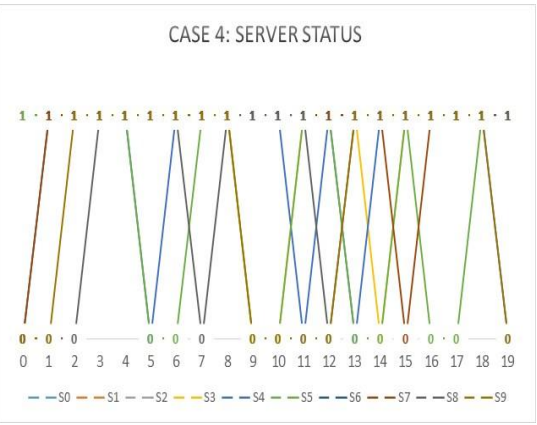
Case 4: Server: 10, Total Job Generated: 0-50 per Interval, Interval: 20, EET: 0-40

The data collected is listed in Table 10. (0 - Idle, 1 - Busy) and shown in Figure 9.

Table 10: Case 4

Interval	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
S0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
S1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
S2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
S3	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0	1	1	1	1	1
S4	1	1	1	1	1	0	1	1	1	1	1	0	1	0	1	1	1	1	1	0
S5	1	1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	0	0	1	1
S6	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
S7	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
S8	0	0	0	1	1	1	1	0	1	1	1	1	0	1	1	1	1	1	1	1
S9	0	0	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	0

Figure 9: Graphical Representation of Case 4



6. CONCLUSIONS

We proposed a framework for load balancing in heterogeneous web server clusters. Load is distributed on the basis of memory and processing requirements. Preliminary evaluation reveals that use of this algorithm is necessary to improve the performance of web servers by proper resource utilization and reducing the mean response time by distributing the workload evenly among the web servers.

7. REFERENCES

[1] Kanungo Priyesh (2013), Scheduling Algorithms in Web Server Clusters, International Journal of Computer Science and Mobile Computing ISSN 2320–088X, IJCSMC, Vol. 2, Issue. 10, October 2013, pg.78 – 85

[2] Kanungo Priyesh (2013), Study of Server Load Balancing Techniques, International Journal of Computer Science & Engineering Technology (IJCSET), ISSN : 2229-3345 Vol. 4 No. 11 Nov 2013

[3] Yousofi Ahmad, Banitaba mostafa and Yazdanpanah Saeed (2011), A Novel Method for Achieving Load Balancing in Web Clusters Based on Congestion Control and Cost Reduction, IEEE Symopsium on Computers & Informatics, IEEE 2011

[4] Mahmood Amjad and Rashid Irfan (2011), Comparison of Load Balancing Algorithms for Clustered Web Servers, Proceedings of the 5th International Conference on IT & Multimedia at UNITEN (ICIMU 2011) Malaysia, 14-16 November 2011

[5] Ho Lai Kuen et al (2004), Improving web server Performance by a clustering-based Dynamic Load Balancing Algorithm, Proceedings of the 18th International Conference on Advanced Information Networking and Application (AINA'04), IEEE 2004

[6] Ling Yibei, Chen Shigang, Lin Xiaola (2003), Towards Better Peformance Measurement of Web Servers, ICICS-PCM, December 2003

[7] Choi Min et al (2003), Improving Performance of Load Balancing System by Using Number of Effective Tasks, Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'03)

[8] Krawczyk Henryk, Urbaniak Arkadiusz (2002), Allocation Strategies of User Requests in Web Server Clusters, Proceedings of the International Conference on Parallel Computing in Electrical Engineering (PARELEC'02)