A Framework for Implementation of Load Balance Access in Computational (Server) Environment

Osah, Stella[1], E. O. Bennett[2] & O. E. Taylor[3]
Department of Computer Science,
Rivers State University,
Port Harcourt,
Nigeria
Stellaattah12@gmail.com[1], bennett.okoni@ust.edu.ng[2],
taylor.onate@ust.edu.ng[3]

Abstract
The computational (server) environment has been faced with several challenges during operations of the cooperating computing systems with adverse outcome. The resultant effect is the gradual slowdown of the performance of the system in attending to user requests, drop in response time, hang and crash issues among others. Some of these issues occur when the server is no longer responding to users request or the server becomes irresponsive to user actions; all these cause the system to fail. Deadlock during computation is also an aspect of the challenges caused on the server as different users compete for resources held by other users. This research presents a framework for implementation of load balance access in a computational (server) environment. This is achieved by identifying factors leading to load imbalance, determining the degree of imbalance, developing a framework that will reduce imbalance. Computing an average neighborhood load balancing using a 10% trigger on the server while checking for situations when the server is overloaded, under loaded and loaded. The average neighborhood was used to achieve this and the system was implemented with visual C#. The system achieved an efficient load balancing process using the average neighborhood load balancing method.

Keywords: *Load Balancing, Computational Environment, Average Neighbourhood Algorithm.*

1. Introduction

In a disseminated computing environment, the term Load Balancing (LB) refers to a process of visibly and intuitively sharing computational resources e.g. network bandwidth, CPU cycles, memory threads, storage space, etc. by routing access requests to certain resources from users or computers at a particular time to specific systems (Wang, 2015). In real world, a workstation user may not use the machine always, however, he might need more than just the machine during active working time. It's important to know that some resources may be heavily loaded, while others are lying idle. Improving on performance is one of the most important issues in circulated systems. The overall performance of the system can often be enhanced to a suitable level just by distributing the workload among the machines. (Kris et al 2015).

How user's job is being processed is an issue of major concern as there is always lots of jobs in queue within the computing situation leaving the systems in a complex scenario. The truth is that every user often desires his jobs processed and finished as quickly as possible, and in events where the computing systems are over loaded, the chances of possible bottlenecks are on the rise while system resources are being allocated. Whenever Load Balancing is implemented within a computing environment, the major aim has always been to deliver high availability of resources (Aditya, 2015). However, the current approaches used in load balancing has too much overhead to contend with, therefore cannot withstand impasses when

too much access demand to same resource are coming in per second and limited resource is available to service the request in queue (Zhao, 2013). This paper presents a framework for implementation of load balance access in a computational environment.

2. Related Literature

In a server environment, load balancing is a framework that is used to share workload evenly across nodes. When used in a computing environment, load balancing guarantees improved performance as well greater user satisfaction and resource usage ratio, ensuring that a server (node) is not overloaded. The outcome enhances overall performance of the machine. When properly implemented, load balancing can help in optimal utilization of the available server resources, hence, minimizes resource consumption rate. It can as well be used as a means of implementing fail-over, allowing for system scalability, get rid of bottlenecks and over-provisioning, leading to reduced response time, etc. (Chana, 2012).

Load balancing is very common in web and database access, Dynamic Name Server (DNS) and name resolution, storages, network bandwidth consumption, and packet routing. With no load balancing in place, chances are very high users will experience serious delays, processing timeouts and possible sustained high latency. Load balancing relates with networking by allowing the deployment of redundant servers to enable spread of communication traffic from services, like Web, DNS, etc. on a pool of servers (Chaczko, 2011). In a situation where two or more distributed computing machines are made to communicate to each other via a network, chances of resource sharing are very high and is feature that will be desired. Additionally, there will be a benefit of performance improvement for the servers as a result of sharing their computational power (CPU), apart from the traditional data, I/O sharing. Load balancing serves as a mechanism which allows job requests to be moved from one server to the other inside the distributed system, this creating a much faster job servicing like minimizing job response time and optimizing resource usage. Several findings have revealed that load balancing between computing devices within distributed system improves the overall performance hugely with corresponding improvement in resource utilization. There are two kinds of load balancing algorithms, namely, static and dynamic.

Load balancing policies in static algorithms are largely centered around information on average system's behavior, while transfer decisions are based on the real current state of the system. The static load balancing patterns rely on prior understanding of applications and statistical data of the system to reach its decisions. Static load balancing algorithm determines the performance of each CPU at the start of execution, afterwards, depending on their levels of performance, computing task is allotted by the controlling (master) CPU. Other CPUs (slaves) then compute the task assigned them and send their results back to the controller. On the other side, static load balancing scheme has its own problem, and that is, once a processor has been selected to handle a task, that decision is final. The selection cannot be altered when a process is being executed to allow for modifications on how the system is loaded (Rajguru, 2000).

However, in dynamic load balancing algorithms, the load is allocated among the processors during the execution time. The controller allots new processes to the slaves using newly gotten information. Dynamic load balancing clearly has the edge over its static counterpart in terms of adjustment. The constant monitoring makes use of CPU cycles, so, care has to be taken by way of when it should be called, because the redistribution introduces additional CPU overhead at runtime (Wang, 2015).

3. Methodology

To successfully achieve the task of load balancing in a computational environment, constructive research and Object-Oriented Analysis and Design was used for this paper.

Constructive research method is an important research procedure in computer science. This approach solves problem through the construction or use of models, diagrams, etc. This research technique is commonly used in operations analysis, mathematics, technical sciences, and clinical medicine and in operations research (Eero et al, 1993). On the other hand, Object – Oriented Analysis and Design when applied leads to object oriented disintegration and is flexible to change with greater level of confidence. Its model of interaction represents interesting entity of the system being modeled. It uses the unified modeling language to represent these models (Booch et al, 2007), (Roebuch, 2011).

3.1 Load Balancer Structure

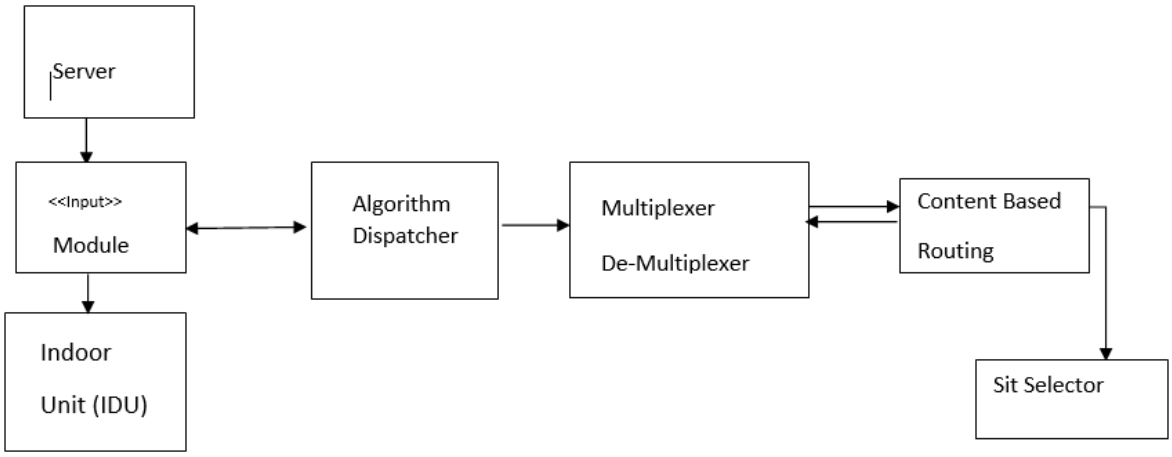

Figure 1 Architecture components for AvNA

This is the system architecture showing were the Average Neighbourhood Algorithm (AvNA) is implemented in the server environment. The signal goes through the input module to the algorithm dispatcher (where AvNA resides). After balancing, the multiplexer and de-multiplexer will convert the signal to a digital base to ensure proper workload balancing is deployed/ distributed to the content base routing before allocation to a particular client/node.

4. Proposed Implementation of Load Balance Access in Computational (Server) Environment

This system consists of three models: the client, the server and the load balancer. The clients are the users of the system while the server is the location from where the client retrieves information. The load balancer interfaces between the client and the server directing client required information to the available server. This load balancer uses the Average Neighbourhood Algorithm (AvNA) to balance the load coming into the server. The proposed AvNA is inspired by the honey bee load balancing algorithm that switches jobs from overloaded Virtual Machines (VM) taking the decision of submitting them to one of the underloaded VMs which is regarded as the destination of the honey bee.

4.1 Average Neighbourhood Algorithm

The proposed framework for load balancing (LB) uses AvNA to distribute workload among available servers to avoid a scenario of servers being overloaded (over-utilized) and under-loaded (under-utilized). It calculates the initial work load and divides by the number of available servers in the computational environment and multiplies the result by the average percentage of the workload. This is achieved when the initial workload is greater than the trigger point of 10% then the LB Algorithm (_AvNA) will be triggered for the load to be

balanced and the balanced average workload will be redistributed among the available servers. Table 1 shows the notations used in the LB algorithm

Table 1. The notations used in the Average Neighbourhood Load Balancing Algorithm

S/N	SYMBOL	NOTATION
1	*Virtual Machines: (VM)*	Average nodes. Nodes operating within a running server environment.
2	*host (i)* *{server}*	The host number i: Number of the overloaded load
3	$PT_{host}(i)$	Processing time of *host(i)*
4	$PT_{VM}(j)$	Processing time of Virtual Machine *(j)*
5	$TL_{host}(i)$	Total length of tasks submitted to *host(i)*
6	$TL_{VM}(j)$	Total length of tasks submitted to *VM(j)*
7	*VM (j)*	The VM number *i*
8	*Sv*	The server
9	PT_{Avg_host}	

The computational environment consists of a set of servers or virtual machines which also contains load balancer and is responsible to find suitable host and machine to allocate task through the AvNA. The following equations were used:

- Average processing time

$$PT_{Avg\_host} = \frac{1}{n}\sum_{i=1}^{n} PT_{host(i)} \quad (1)$$

- Percentage Load Balancing

$$\%ALB = \frac{Sv - initial\ load}{Load\ Average}\ x\ 100\% \quad (2)$$

Average Neighbourhood Algorithm

Calculate Average PT of VMs
Ave PT (VMs) = $PT_{host(i)}$
If PT (VM_j) > PT_{host} (i) // VM is Overloaded
// calculated degree of imbalance and redistribute workload {length}
$TL_{VM}(J)$ - $TL_{host(i)}$
If PT(VMj) < PT(i) // VM is Underloaded
// Calculate Degree of imbalance
$TL_{host(i)}$ - $TL_{Mv(j)}$ ///The degree of Imbalance factor
If $PT_{(VMj)}$ = PT(i) // VM = Balanced

The AvNA presents detailed explanation of the system, when the load is overloaded, underloaded and balanced.
The load is said to be overloaded when the processing time *(PT)* scheduled by the virtual machine is less than the processing time of the host system in the computing environment. When the processing time of the virtual machine is greater than the processing time of the host machine the server is underloaded. To calculate the degree of imbalance, the total amount of workload in the host system is subtracted from the virtual machine in the server. The load becomes underloaded when the processing time of the virtual machine is less than the processing time of host system. While, when the load is imbalance, the total length of the task submitted in *host (i)* is subtracted from the total length of the task in the virtual machine *(vm)*.

The balances the system workload on the server by calculating the average percentage of the workload in the server environment. It calculates the workload to be balanced by checking the neighbors load before redistribution of workload: if the server is overloaded or underloaded that means the incoming load is greater or lesser than the threshold value of trigger. This algorithm can only calculate and redistribute when the load is 10% greater than or less than the average workload within the cooperating computing nodes.

5. Results and Evaluation

The simulation result shows the distribution stage of the system, the initial volume of the system, the percentage at which the Average Neighbourhood Algorithm will be triggered to start the balancing and the actual balancing of the system. The system considered 4 and 8 server scenario and figure 3 and 4 show the output of the system.

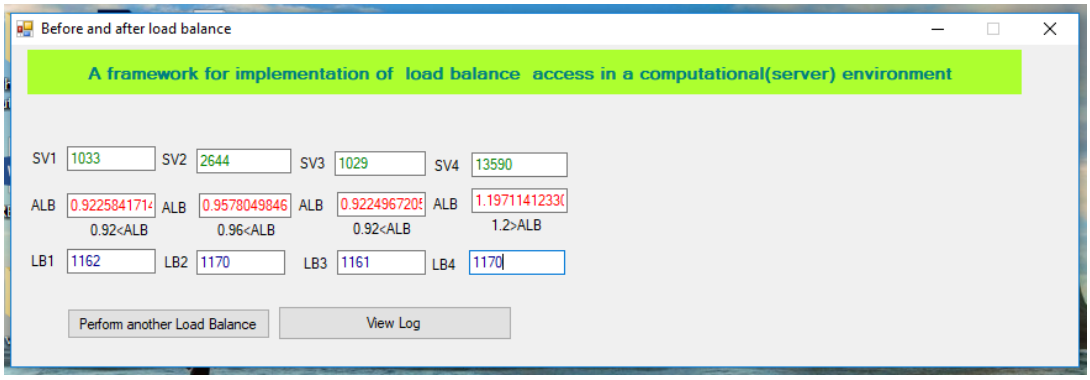

Figure 3: Output result for 4 servers after load balancing

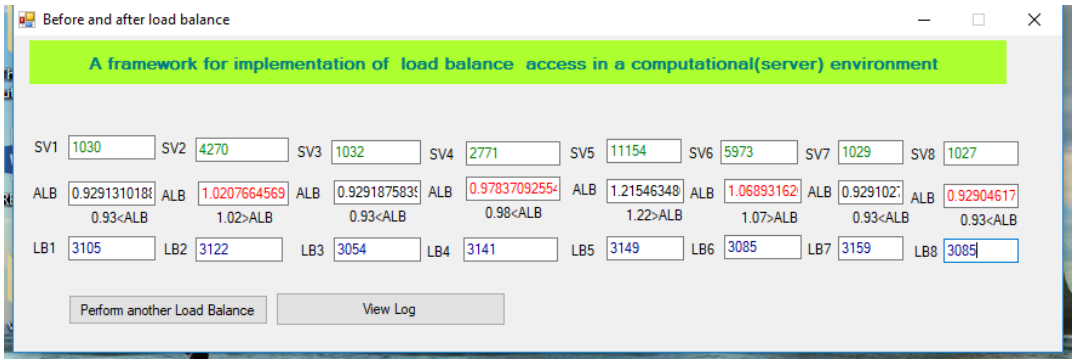

Figure 4: Output result for 8 servers after load balancing

Table 1 Result table for 4 - servers

No of servers	**Sv1**	**Sv2**	**Sv3**	**Sv4**
Initial loads	1026	4676	1038	10386
	1032	5283	1027	12879
	1030	4179	1031	12244
	1033	2644	1029	13590
%ALB	1.23	1.03	1.23	1.26
Actual LAB	1267	1481	3359	3979

Table 2 Result table for 8 - servers

No of server	Sv1	Sv2	Sv3	Sv4	Sv5	Sv6	Sv7	Sv8
Initial loads	1030	4270	1032	2771	11154	5973	1029	1027
	1038	3311	1031	2991	10568	4291	1035	1027
	1031	3132	1029	2746	10116	4589	1029	1035
	1028	2156	1038	3667	13159	4827	1034	1025
%ALB	1.28	1.17	1.28	1.35	1.34	1.27	1.28	1.28
Actual LAB	2673	1134	3351	1745	3624	1638	3423	3417

Graphical Representation

The graph shows the time and the load representation of load balancing system.

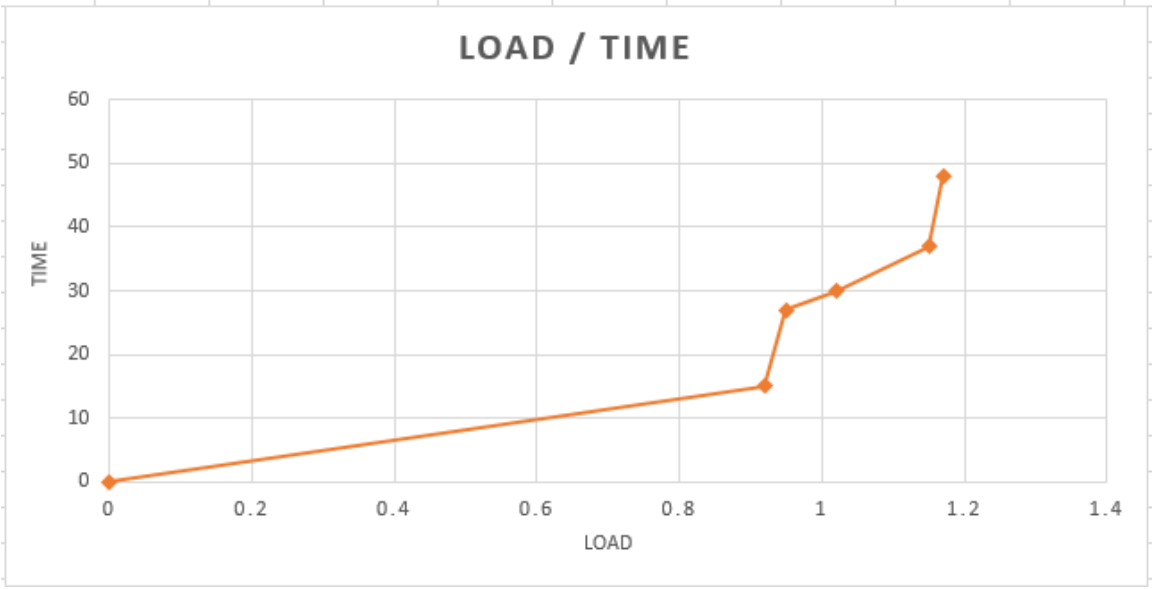

Figure 5: Degree of Load Balancing Response Time

Figure 5 is the average response time varies among different simulation; from the graphical illustration, the time required for each load shows a clear separation from each other.

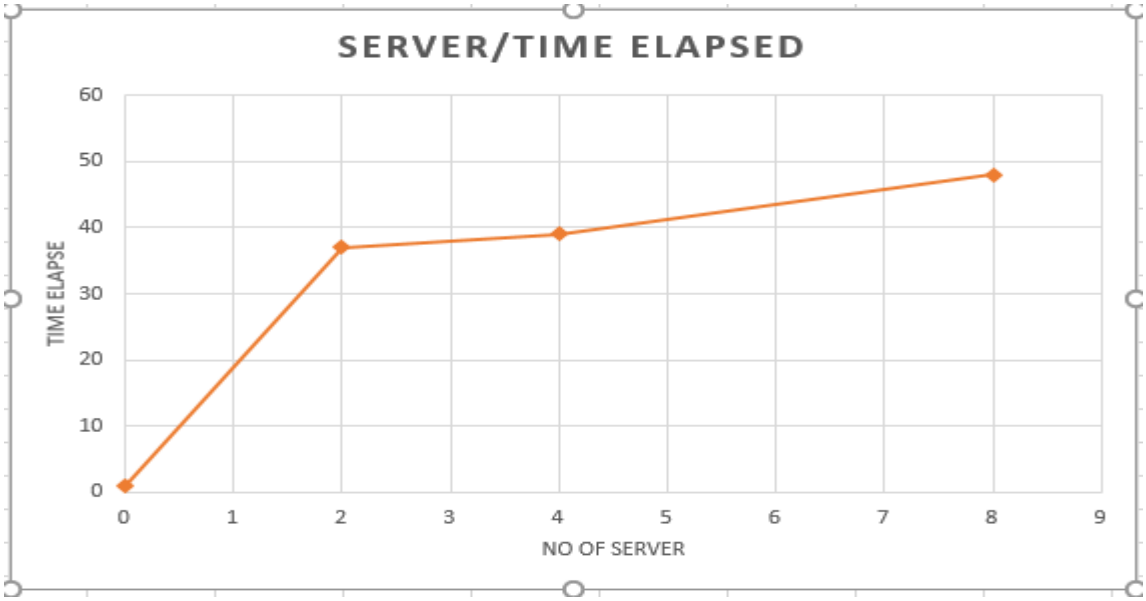

Figure 6: Server against time elapsed

Figure 6 shows number of servers against the time it takes to complete a load balancing process. It is clear that, the number of servers also affects the time the system takes to balance or redistribute workload among cooperating nodes in the system.

6. Discussion of Results

From the results observed, computing environment with systems numbering 4 and 8 servers where selected and the average neighborhood algorithm was executed during the system run. Figure 3 and figure 4 shows the output results for 4 and 8 servers after load balancing with sv1, sv2, sv3 and sv4 showing the initial distributed workload in figure 3 and LB1, LB2, LB3, LB4 showing the output of the balanced load.

In figure 4, the servers sv1, sv2, sv3, sv4, sv5, sv6, sv7, and sv8 show automatically generated initial loads for each node and the corresponding balanced load is shown in servers LB1 to LB8 respectively.

From figure 3 and 4, the initial workload generated shows the unbalanced state of the servers. After applying the load balancing algorithm, the results showing on both figures presents an evenly distributed loads among the servers.

Table 1 and Table 2 are the summary of the simulation results for 4 servers and 8 servers at different execution time. Figure 5 represents the load and time evaluation which gives the degree of load balancing response time.

Figure 6 depicts the number of servers and time taken to complete the redistribution process.

Conclusion

Load balance access removes the overloaded workload and provides equal and approximate service in multi access / distributed computing environment. The load balancing method can be used for the better utilization and understanding of load balancing systems. This research observes that load balancing is important issue in a computing environment as it balances the storage and service demands in data center etc. It also helps in answering the question of how to achieve minimum overhead on the server system and maximum resource utilization, throughput, how to reduce traffic and get better performance of the system.

References

Boettcher, S., & Percus, A. G. (1999). Extremal optimization: Methods derived from co-evolution. Paper presented at the Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1.

Chana, Y. and N. J. Navimipour (2012). "Online knowledge sharing mechanisms: a systematic review of the state of the art literature and recommendations for future research." Information Systems Frontiers: 1-21.

Chaczko, A., et al. (2011). "Virtual machine provisioning through satellite communications in federated Cloud environments." Future Generation Computer Systems 28(1): 85-93.

Charband, Y., & Navimipour, N. J. (2016). Online knowledge sharing mechanisms: a systematic review of the state of the art literature and recommendations for future research. Information Systems Frontiers, 18(6), 1131-1151.

Daraghmi, E. Y., & Yuan, S.-M. (2015). A small world based overlay network for improving dynamic load-balancing. Journal of Systems and Software, 107, 187-203.

Eero, D. C. (2000). "Cloud computing: A value creation model." Computer Standards & Interfaces 38: 72- 77.

Gao, R., & Wu, J. (2015). Dynamic load balancing strategy for cloud computing with ant colony optimization. Future Internet, 7(4), 465-483.

Gopinath, P. G., & Vasudevan, S. K. (2015). An in-depth analysis and study of Load balancing techniques in the cloud computing environment. Procedia Computer Science, 50, 427-432.

Krishna, P. V. (2013). Honey bee behavior inspired load balancing of tasks in cloud computing environments. Applied Soft Computing, 13(5), 2292-2303.

Kumar, S., & Dutta, K. (2016). Securing mobile ad hoc networks: Challenges and solutions. International Journal of Handheld Computing Research (IJHCR), 7(1), 26-76.

Milani, A. S., & Navimipour, N. J. (2016). Load balancing mechanisms and techniques in the cloud environments: Systematic literature review and future trends. Journal of Network and Computer Applications, 71, 86-98.

Ren, X., Lin, R., & Zou, H. (2011). A dynamic load balancing strategy for cloud computing platform based on exponential smoothing forecast. Paper presented at the 2011 IEEE International Conference on Cloud Computing and Intelligence Systems.

Samanta, P., & Mondal, R. K. (2016). Load balancing through arranging task with completion time. International Journal Of Grid And Distributed Computing, 9(5), 273-282.

Wang, S.-C., Yan, K.-Q., Wang, S.-S., & Chen, C.-W. (2011). A three-phases scheduling in a hierarchical cloud computing network. Paper presented at the 2011 Third International Conference on Communications and Mobile Computing.

Zhang, J., Huang, H., & Wang, X. (2016). Resource provision algorithms in cloud computing: A survey. Journal of Network and Computer Applications, 64, 23-42.