Application of Fuzzy Logic in Load Balancing of Homogenous Distributed Systems[1]

Ali M. Alakeel *alakeel@ut.edu.sa*
Department of Computer Science
Faculty of Computing and Information Technology
University of Tabuk
P.O.Box 741, Tabuk 71491, Saudi Arabia

Abstract

Various studies have shown that distributing the work load evenly among processors of a distributed system highly improves system performance and increases resource utilization. This process is known as load balancing. Fuzzy logic has been applied in many fields of science and industry to deal with uncertainties. Existing research in using fuzzy logic for the purpose of load balancing has only concentrated in utilizing fuzzy logic concepts in describing processors load and tasks execution length. The responsibility of the fuzzy-based load balancing process itself, however, has not been discussed and in most reported work is assumed to be performed in a distributed fashion by all nodes in the network. This paper proposes a new fuzzy dynamic load balancing algorithm for homogenous distributed systems. The proposed algorithm utilizes fuzzy logic in dealing with inaccurate load information, making load distribution decisions, and maintaining overall system stability. In terms of control, we propose a new approach that specifies how, when, and by which node the load balancing is implemented. Our approach is called Centralized-But-Distributed (CBD). An evaluation study of the proposed algorithm shows that our algorithm is able to reduce the average response time and average queue length as compared to known load balancing algorithms reported in the literature.

Keywords: Dynamic Load Balancing, Fuzzy Logic, Distributed System, Algorithms.

1. INTRODUCTION

Various studies have shown that distributing the work load evenly among processors of a distributed system highly improves system performance and increases resource utilization. Load balancing in distributed computer systems may be defined as the process of redistributing the work load among processors in the system to improve system performance [1]. Dynamic load balancing algorithms monitor changes on the system work load and redistribute the work load accordingly, e.g., [1]-[13]. A dynamic load balancing algorithm is usually composed of three strategies: transfer strategy, location strategy, and information strategy. Transfer strategy decides on which tasks are eligible for transfer to other nodes for processing. Location strategy nominates a remote node to execute a transferred task. Information strategy is the information center of a load balancing algorithm. It is responsible for providing location and transfer strategies at each node with the necessary information required to take their decisions. Information strategy is an important part of a load balancing algorithm. The worth and complexity of any dynamic load balancing algorithm depends heavily on the performance of its information strategy. The implementation responsibility or control of a dynamic load balancing algorithm can take three different forms: centralized, distributed, or semi-distributed. In a centralized load distribution algorithm, a single node (called central node) in the network is nominated to be responsible for all load distribution in the network. In a distributed load balancing algorithm, the responsibility is distributed where each node in the network carries an equal share of the responsibility and

[1] An earlier version of this paper, [32], was presented at the International Conference on Computer and Information Technology, Zurich, Switzerland, January 15-17, 2012.

executes the same algorithm. In a semi-distributed load balancing algorithm, the network is segmented into clusters where each cluster contains a set of nodes. The control within each cluster is centralized, i.e. a central node is nominated to take charge of the load balancing process within its set. In the semi-distributed approach, load balancing of the whole distributed system is achieved through the cooperation of central nodes of each cluster, i.e. the responsibility is distributed among the central nodes of each cluster.

This paper proposes a new fuzzy dynamic load balancing algorithm for homogenous distributed systems. The proposed algorithm utilizes fuzzy logic, e.g., [29]-[31], in dealing with inaccurate load information, making load distribution decisions, and maintaining overall system stability. In terms of control, we propose a new approach that specifies how, when, and by which node the load balancing is implemented. Our approach is called Centralized-But-Distributed (CBD). Also we introduce a new location policy which utilizes Fuzzy Logic techniques in redistributing the load from heavily loaded nodes to lightly loaded nodes in the system. Our load balancing algorithm includes an explicit mechanism for monitoring and tuning system stability. System stability is maintained through the dynamic adjustment and tuning of various parameters incorporated in the algorithm.

The rest of this paper is organized as follows. Section 2 provides an overview dynamic load balancing algorithms. Section 3 presents our proposed fuzzy dynamic load balancing algorithm. An experimentation study of the proposed algorithm is presented in Section 4 and in Section 5, we discuss our conclusions and future research.

2. AN OVERVIEW OF DYNAMIC LOAD BALANCING ALGORITHMS

This section provides an overview of dynamic load balancing in distributed systems with more emphasis on the components related to our work presented in this paper. Specifically, in subsection 2.1, different approaches of controlling the load balancing process is presented and subsection 2.2 presents different load balancing strategies that we used in our evaluation study to compare the performance of the proposed algorithm.

2.1 The Controlling Mechanism

Along with various load balancing strategies which may be applied independently or tailored to enhance the performance of an algorithm for solving a certain problem, different policies of where to put the control of the load balancing algorithm have been proposed in the literature: centralized, distributed, or semi-distributed.

A centralized load balancing strategy assigns a single processor the responsibility of initiating and monitoring the load balance operation. In this strategy, a dedicated processor gathers the global information about the state of the system and assigns tasks to individual processors. Despite its high potential of achieving optimal performance, centralized strategies have some disadvantages: high vulnerability to failures, storage requirements for maintaining the state information - especially for large systems, and the dependability of the performance of the system on the central processor which could result in a bottleneck [1].

In a distributed load balancing strategy, each processor executes the same algorithm and exchanges information with other processors about the state of the system. Each processor may send or receive work on the basis of a sender-initiated or a receiver-initiated policy. In a sender-initiated policy, the sender decides which job gets sent to which receiver. In a receiver-initiated policy, the receiver searches for more work to do. Intuitively, queues are formed at senders if a receiver-initiative policy is used, while they are formed at receivers if a sender-initiative policy is used. Additionally, scheduling decisions are made when a new job arrives at the sender in a sender-initiative, while they are made at the departure of a job in a receiver-initiative policy. The determination of which policy is adopted depends upon the load transfer request which can be initiated by an over-loaded or under-loaded processor [1], [3], [6], [7]. Assuming that process migration cost under the two strategies is comparable, it has been demonstrated in, [4], [6], and

[8], using analytical models and simulations, that sender-initiated strategies generally perform better at lower system loads while receiver-initiated strategies perform better at higher system loads. Some of the advantages offered by the distributed policy are: Fault tolerance, minimum storage requirements to keep status information, and the availability of system state information at all nodes. The distributed policy still has some disadvantage, one of which is that optimal scheduling decisions are difficult to make because of the rapidly changing environment introduced by the arrivals and departures from individual processors. Another disadvantage is the extra communication overhead introduced by all processors trying to gather information about each other. To mitigate this overhead, some distributed strategies minimize the amount of information exchanged, which has a negative reflection on the performance of an algorithm.

The semi-distributed policy comes in the middle between centralized and distributed policies. It is introduced to take the best of each and to avoid the major drawbacks of each of the two policies. The semi-distributed strategy is based on the partitioning of the processors into equal sized sets. Each set adopts a centralized policy where a central processor takes charge of load balancing within its set. The sets together adopt a distributed policy where each central processor of each set exchanges information with other central processors of other sets to achieve a global load balance.

It has been shown in [1], that the semi-distributed policy produces a better performance than the centralized and distributed policies. Research demonstrates that each central processor yields optimal load balance locally within its set. Moreover, this policy does not incur high communication overhead while gathering system state information. Although this policy is a mediator between the centralized and the distributed ones, it fits large distributed systems better than small systems.

2.2 Different Load Balance Strategies

This section identifies some, not necessarily all or the best, dynamic load balancing strategies that have been reported in literature. It has to be emphasized that the selection of these specific strategies is to serve our presentation only and not to be interpreted as an exhaustive selection or classification.

2.2.1 The Random Strategy

Under a random load balancing strategy, a local node selects a remote node randomly and transfers the job there for execution [5]. Upon receiving this job, the remote node executes it if its load, i.e., queue length, is below a predefined threshold. Otherwise, this remote node will select a new destination node for this job. To avoid having this job ping ponged among nodes without getting serviced, a limit on the number of hops it could take is imposed which enforces the last node to receive that job to execute the job when this limit is reached regardless of its current load. As shown in [5], the performance of this simple strategy, which does not employ any information in its selection, was significant as compared to a system with no load balancing at all. The performance of this strategy is usually used as a reference point to compare other load balancing algorithms that collect global information.

2.2.2 The Threshold Strategy

In this type of load balancing algorithms, a threshold value T is used to decide whether a task is executed locally or remotely. The threshold is the processor's queue length. The queue length is the number of processes in service plus the number of processes waiting in the queue. Threshold value assumes static value in an implementation of the algorithm [3] and [6]. In this strategy, a processor will try to execute a newly arriving task locally unless this processor threshold has been reached. In this case, this processor will select another processor randomly and probe it to determine if transferring the task to the probed processor will place it above the threshold or not. If it does not, then the task is transferred to the probed processor which has to execute the task there without any attempts to transfer it to a third one. If it does place it above the threshold, then another processor is selected in the same manner. This operation continues up to a certain limit called the probing limit. After that, if no destination processor is found, the task is executed locally

[6]. It should be noted that the threshold strategy requires no exchange of information among processors in order to decide whether to transfer a task or not [6]. This is an advantage because it minimizes communication overhead. Another advantage is that the threshold policy avoids extra transfer of tasks to processors that are above the threshold. It has been shown, in [6], that the threshold algorithm with T=1 yields the best performance for low to moderate system load, and the threshold algorithm with T=2 gives the best performance for high system load.

2.2.3 The Greedy Strategy

This strategy, the current state of each processor P represented by a function *f*(n) , where n is the number of tasks currently at the processor. If a task arrives at P and number of tasks n is greater than zero, then this processor looks for a remote processor that has its state less than or equal to *f*(n). If a remote processor is found with this property, then the task is transferred there. The performance of this strategy depends on the selection of the function f(n). It has been shown in [33] that *f*(n) < n must holds in order to achieve good performance. Also, n-1, n div 2, n div 3, n div 4, etc. are possible values for *f*(n). Furthermore, it has been shown in [33] that *f*(n) = n div 3 yields the best results and that the greedy strategy outperforms the threshold strategy with T=1 in all experiments. The greedy strategy adopts a cyclic probing mechanism instead of the random selection used in the threshold strategy. In this cyclic probing mechanism, processor i probes processor (i+j) mod N, N representing the number of processors in the system, in the j^{th} probe to locate a suitable destination processor. For example, in a system with 5 processors numbered 0,1,2,3, and 4 respectively, Processor 1 will first probe processor 2. If this attempt is not successful, it will probe 3 and so on. As in the threshold strategy, once a task is transferred to a remote processor it must be executed there [33]. Despite the similarities between the two strategies, it has been demonstrated using simulation results in [33] that the greedy strategy outperforms the threshold strategy. This improvement is attributed to the fact that the greedy strategy attempts to transfer every task that arrives at a busy processor whereas the threshold strategy attempts to transfer only when a task arrives at a processor which has reached the threshold T or higher.

2.2.4 The Shortest Queue Strategy

The shortest load balancing strategy selects a subset of remote nodes randomly and probes them to find out their current load, i.e. queue length. The remote node with the smallest queue length is then selected. If this selected node's queue length is smaller than a certain threshold, then the job is transferred there, otherwise the job is executed locally. Although the shortest strategy attempts to make a wiser selection than the threshold does, it is shown in [5] that there was not a significant gain in performance over what has been achieved by the threshold strategy.

2.2.5 The Bidding Strategy

The basic idea of this strategy is bids. The overloaded processor looking for help in executing some of its tasks requests other processors to submit their bids. Bid information includes the current work load status about each processor. After receiving all bids from participating processors, the original processor selects to whom it will send some of its tasks for execution. A major drawback of this strategy is the possibility that one processor will become overloaded as a result of its winning many bids. To overcome this problem, some variations of this strategy would allow the bidder processor to accept or reject the tasks sent by the original processor. This could be done by allowing it to send a message to the original processor informing it of whether the work has been accepted or rejected. Since a processor's load could change while these messages take place, the final selection might not turn out to be as good as it seems to be at earlier time or vice versa [7]. Different algorithms have been proposed in the literature to determine who gets to initiate the bid, bid information, bid selection, bid participation, and bid evaluation [1], [3], [4], [5], [7]. The performance of this strategy depends on the amount of information exchanged, the selection of bids, and communication overhead [3]. More information exchange enhances the performance and provides a stronger basis for selection but also requires extra communication overhead [3], [7], [33].

2.2.6 The Drafting Strategy

The drafting strategy differs from the bidding strategy in the way it allows process migration and in the manner it attempts to achieve load balance. The drafting policy tries to alleviate some of the communication overhead introduced by the bidding strategy. The drafting policy achieves load balance by keeping all processors busy rather than evenly distributing the work load among participating processors (which is one of the objectives of the bidding strategy). In the bidding strategy, to keep all processors evenly loaded, groups of processes will be required to migrate from a heavily loaded processor to a lightly loaded processor. Consequently, it is possible to find some of these processes migrating back as a result of the unpredictable change of the processor's work load. To allow for this problem in the bidding approach, the drafting strategy allows only one process to migrate at a time rather than group migration [7].

The drafting strategy adopts a process migration policy which is based on giving the control to the lightly loaded processors. Lightly loaded processors initiate a process migration instead of having process migration being triggered by a processor being overloaded as in the bidding strategy [7]. In drafting, the number of processes currently at the processor is used for work load evaluation. Each processor maintains its work load and identifies itself as in one of the following states: H-Load, N-Load, or L-load. An H-Load, heavy load, indicates that some of this processor's processes can migrate to other processors. An N-Load, normal load, indicates that there is no intention for process migration. A L-Load, light-load, indicates that this processor is willing to accept some migrant processes. A load table is used at each processor to hold this information about others processors and act as a billboard from which the global information of the system is obtained. When a load change occurs in a processor, it will broadcast its load information to other nodes so that they will update their load tables.

When the processor becomes lightly loaded, i.e. L-Load, it will identify other processors having the status of H-Load from its load table and send them a draft-request message. This message indicates that the drafting processor is willing to accept more work. If by the time it receives this message it is still in H-Load, each remote (drafted) processor will respond by sending a draft-respond message which contains draft-age information. Otherwise the current load status will be returned to the drafting processor, adopting the concept that a process is allowed to migrate only if it is expecting a better response time and age is associated with each draftable process. Some of the parameters that may be used for age determination are: Process waiting time, process priority, or process arrival [7]. The draft-age is determined by the ages of those processes nominated to be drafted. Various alternatives for draft-age calculations are possible. The selection of the draft age to be the maximum age of all draftable processes, the average age of the draftable processes, or simply the number of draftable processes are some of them [7]. When all draft-response messages are received, the drafting processor calculates draft-standard criteria. Draft-standard criteria are calculated based on the draft-ages received and used to ensure fairness of selection among drafted processes. The choice of draft standard is crucial to the performance of this strategy and is determined at the system design stage. After calculating the draft-standard, a draft-select message is sent to the drafted processor that has the highest draft-age. The drafting processor will send the draft-select message, only if it is still on the L-Load state, otherwise it will not accept any migrating processes.

3. THE PROPOSED FUZZY LOAD BALANCING ALGORITHM

In this section, we present our proposed load balancing algorithm for homogeneous distributed computer systems that is based on fuzzy logic capabilities.

3.1 A Background

In our daily life, we use words and terms, which are vague or fuzzy such as:
“The server is *slow*” or
“The weather is *hot*” or
“John is *tall.*”

Fuzzy Logic [29], [31], gives us the ability to quantify and reason with words which have ambiguous meanings such the words above. In fuzzy sets [29], an object may belong partially to a set as opposed to classical or "crisp" sets in which an object may belong to a set or not. For example, in a universe of heights (in feet) for adult people defined as:
μ= {5, 5.5, 6, 6.5, 7, 7.5, 8}, a fuzzy subset TALL can be defined as follows:

TALL = [0/5, .125/5.5, .5/6, .875/6.5, 1/7, 1/7.5, 1/8].

In this simple example, the degree of membership for the members of the universe μ, with respect to the set TALL may be interpreted as that the value "6" belongs to the set TALL 60% percent of the time while the value 8 belongs to the set TALL all the time.

3.2 The System Model

In this presentation, we assume the following system model. We assume N, where N > 1, independent homogenous nodes are connected by a local area network where each node consisting of a single processor. A single typed tasks arrive to a node could either come from outside the network or from other nodes in the network. We assume that all nodes are subjected to the same average arrival rate of tasks coming from outside the network. All tasks are queued at each node and are served on a First Come First Serve (FCFS) basis.

3.3 The Assumptions

Our presentation is based on the following assumptions:

- Nodes of the distributed system are numbered from 1 to N, where N is the total number of nodes in the system. The numerical number of each node represents its identification (ID) in the system.
- Nodes are connected by a broadcast network and the cost of sending a message between any two nodes is the same.
- We assume that each processor is in some state S_k where it has a number of tasks T_k. Furthermore, we assume that the distributed system is initially in a steady state.
- Given this configuration, the load balancer starts and tries to examine the overall state of the system and takes the necessary corrective actions accordingly based on the objectives this algorithm assumes.

3.4 The Objectives

In this presentation, once a node is elected, by the algorithm, to act as the load balancer of the distributed system under consideration, this node will have the following objectives:

1) To obtain full information about the load of each host in the system efficiently.
2) To maintain a load balance among the distributed system hosts within a difference *d.* The value of *d* varies during the operation of the load balancing algorithm and adjusts dynamically taking into account the current state of the system and the communication costs. The proper range of *d* will only be determined after experimentations with this algorithm because this process has to be done efficiently in terms of communication time required.
3) To select the most appropriate time to launch the load balancing process. Since keeping the load balancer working at all times is a burden on the system, we propose an efficient way of triggering the load balancer, which is described as follows. The load balancer algorithm is triggered when one of the following conditions is satisfied:
    a) When a node becomes idle, or below a threshold value.
    b) When the load of a node exceeds a threshold value.
4) To ensure that only one node is working as the load balancer at a time. It is possible that more than one node can meet either one of the above conditions. This would cause multiple load balancers to be active at the same time. To prevent this, our algorithm provides a mechanism which ensures that only one load balancer is active at a given time.

3.5 The Algorithm Steps

According to the proposed fuzzy based load balancing algorithm, the load balancer node performs the following steps:

1) Obtain the current load of the distributed system. This is accomplished by broadcasting a request for status message to all nodes in the system.
2) Upon receiving the response from all nodes, the load balancer assigns each node of the system including itself a fuzzy value in the interval [0,1] which represents the load of that node relative to the overall load of the distributed system. This assignment is achieved by forming a fuzzy set, LOADED = {light, normal, heavy}, that represents the load of the system. Using Fuzzy Logic techniques each node of the distributed system is assigned a membership value depending on its current load. The membership value is in the interval [0,1] and reflects the compatibility of the load at a specific node to the fuzzy term LOADED which is represented by a fuzzy set. The assignment of membership values (grades) is based on the S-function [31] which is shown in Fig. 1. S-functions may be described as follows.
    - A mathematical function that is used in fuzzy sets as a membership function.
    - A simple but valuable tool in defining fuzzy functions such as the word "load."
    - The symbols α, β, and γ are parameters which may be adjusted to fit the desired membership data.
    - The S-function is flat at a value of zero for $\times \leq \alpha$ and at 1 for $\times \geq \gamma$. In between α and γ, the S-function is a quadratic function of $\times$. The β parameter corresponds to the crossover point of 0.5 and is $(\alpha + \gamma) / 2$.
3) Using the results of step (2), the load balancer categorizes all nodes into three separate groups: Underloaded, Normal, and Overloaded. Where Normal nodes do not need any load balancing and will be left unchanged. Both Underloaded and Overloaded nodes need help and will be the target of the load balancing process.
4) Create a mapping from Overloaded nodes to Underloaded nodes. The outcome of this mapping tells each overloaded node where to ship some of the extra work it has. As a result of this mapping, the load balancer sends each overloaded node a message specifying the ID of each possible underloaded node and the number of tasks the overloaded node should ship to the underloaded node. This information is formed in a list arranged as: (ID_1, #tasks), (ID_2, #tasks),…, (ID_N, #tasks). To perform this mapping, we adopt Evans et al. [31] probability model by using the load at each node and compute the probability of sending a task from an overloaded node *i* to and underloaded node *j*.

3.6 An Illustrative Example

In this section, we present a simplified example that shows how system load may be dealt with using fuzzy logic in our proposed algorithm.

- Assume a distributed system with 4 nodes, labeled as N1, N2, N3 and N4.
- Assume that at a time *t*, the average queue length in the system is 15 jobs.
- Let μ = {1, 2, 3, 4, …, 15} be the universe of jobs.
- To describe our system load using fuzzy logic, we can create three different fuzzy sets [29] as follows:
    - lightly-loaded = {1/1, .75/2, .5/3, .25/4} from the codomain μ_l ={1, 2, 3, 4}
    - normally-loaded= {1/4, .75/5, .5/6, .25/7} from the codomain μ_n ={4, 5, 6, 7}, and
    - heavily-loaded = { .2/7, .3/8, .4/9, .5/10, .6/11, .7/12, .8/13, .9/14, 1/15} from the codomain μ_h ={7, 8, 9, 10, 11, 12, 13, 14, 15}.

- Assume that at a time t, the load at each node is found to be as follows:
    - N1 (8 jobs), N2 (5 jobs), N3 (11 jobs) and N4 (2 jobs).
- A possible mapping of nodes to the three fuzzy sets is:
    - L_load = {0/1, .5/2, 0/3, .9/4},
    - N_load = {.125/1, .75/2, 0/3, .25/4}, and

- H_load = {0.5/1, 0/2, .875/3, 0/4}.

It should be noted that the main contribution of our proposed algorithm is in the part which is responsible for selecting the best time to trigger the load balancing process and in performing the load balancing process itself. In this regard, a copy of the load balancing algorithm resides on each host as in the distributed load balancing algorithm reported in the literature. In our approach, the responsibility of load balancing is centralized in action, but distributed in time. This means that only a single node will be responsible for performing the load balancing processes of the distributed system. This node, however, is not the only one who will complete this task all of the time, as is in the traditional centralized approach. Instead, each node might have the chance to perform the load balancing task. The selection of which node actually does the work is dynamically determined, however. By doing this, our approach attempts to get the best of the well-known centralized and distributed approaches to the load balancing problem.

$$S(X;\alpha,\beta,\gamma)=\begin{cases} 0 & for \quad X\le\alpha \\ 2\left(\frac{X-\alpha}{\gamma-\alpha}\right)^2 & for \ \alpha\le X\le\beta \\ 1-2\left(\frac{X-\gamma}{\gamma-\alpha}\right)^2 & for \ \beta 2\left(\frac{X-\alpha}{\gamma-\alpha}\right)^2 \\ 1 & for \quad X\ge\gamma \end{cases}$$

FIGURE 1: The S-function.

4. AN EXPERIMENTAL STUDY

The In order to evaluate the effectiveness of our proposed fuzzy load balancing algorithm for homogeneous distributed systems, we have conducted an evaluation study using simulation. In this experiment, we have assumed that job's inter-arrival time follows the exponential distribution. Also, we have assumed that job's execution time is drawn from the uniform distribution. Furthermore, we have assumed a fixed communication time between each two hops in the network. In our experiment, we have used a network composed of four nodes of the same configuration. In this study, a node may get load (jobs) from outside the network or from inside the network because a job inside the network is allowed to move from one node to another for a service. During the period of the experiment, we have subjected each node to different surges of requests from outside of the network.

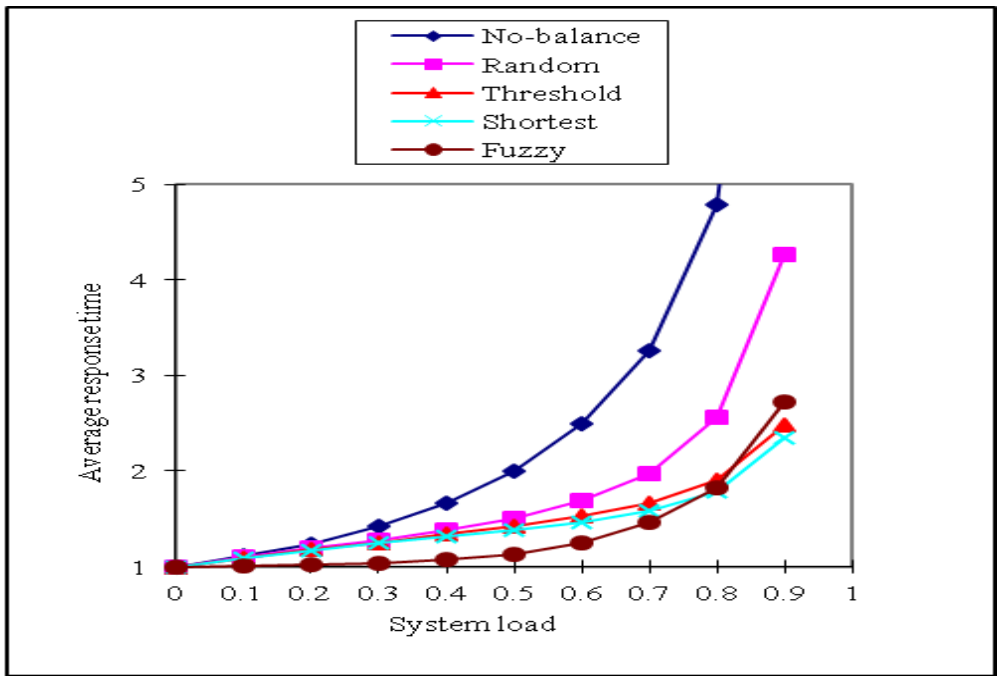

FIGURE 2: The Performance comparison based on average response time.

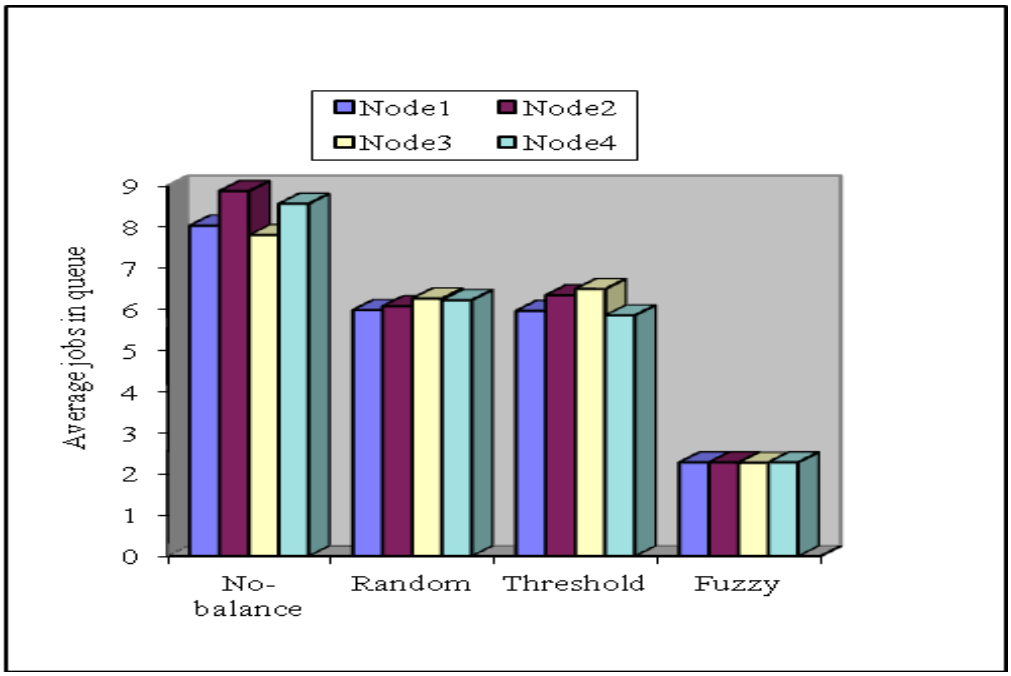

FIGURE 3: The Performance comparison based on average queue length.

At a certain periods of time, the load balancing algorithm is triggered as described in Sec 3. Based on the decisions made by the load balancing algorithm, the load (jobs) may move from node to another in order to load-balance the network. In this stage of our evaluation study, we have measured two major performance indicators of distributed systems, namely, the average response time and the average processor's queue length. We have compared our algorithm to a well-known load balancing strategies from those discussed in Sec. 2. In Fig. 2, the x-axis projects

the system load represented by a fuzzy set as described in Sec. 3, while the y-axis reflects the system's average response time in milliseconds.

As shown in Fig. 2, the proposed fuzzy load balancing algorithm, "Fuzzy," was able to reduce the average response time as compared to Threshold, Shortest and Random load balancing strategies. In Fig. 3, the x-axis projects the average queue length at each processor in our system, while the y-axis reflects the system's average response time in milliseconds. As shown in Fig. 3, the proposed algorithm was also able to reduce job's queue length as compared to the Random and Threshold strategies. Although the results of this experiment are encouraging, the proposed algorithm still needs to be evaluated against different systems load and larger distributed system with thousands of processors.

5. CONCLUSIONS AND FUTURE RESEARCH

This paper proposes a new dynamic load balancing algorithm for homogenous distributed computer systems which employs fuzzy logic in dealing with inaccurate load information, making load distribution decisions, and maintaining overall system stability. The main contribution of our proposal is in the part which is responsible for selecting the best time to trigger the load balancing process and in performing the load balancing process itself. In order to evaluate the proposed algorithm, an overvaluation study was conducted using simulation. The result of this experimental study shows that the proposed fuzzy load balancing algorithm was able to produce a promising system's performance improvements as compared to well-known balancing algorithms reported in the literature. For future research, it is imperative to conduct more experiments that evaluate the performance of the proposed model in computer networks consisting of thousands of nodes. Additionally, it is our intention to construct a theoretical model for our proposed algorithm and compare the results reached with our simulation results presented in this paper.

6. REFERENCES

[1] I. Ahmed and A. Ghafoor, "Semi-Distributed Load Balancing for Massively Parallel Multicomputers," IEEE Trans. Software Eng., vol. 17, no. 10, pp 987-1004, October 1991.

[2] T. L. Casavant, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems," IEEE Trans. Software Eng., vol 14, no. 2, pp 141-154, February 1988.

[3] Y. Wang and R. Morris, "Load Sharing in Distributed Systems," IEEE Trans. Comput., vol. C-34, no. 3, pp. 204-217, Mar. 1985.

[4] K. Ramamritham, J. A. Stankovic, and W. Zhao, "Distributed Scheduling of Tasks with Deadlines and Resource Requirements," IEEE Trans. Comput., vol. 38, no. 8, pp 1110-1123, August 1989.

[5] J. A. Stankovic, K. Ramamritham, and S. Cheng, "Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems," IEEE Trans. Comput., vol. C-34, no. 12, pp. 1130-1143, December 1985.

[6] D.L. Eager, E.D. Lazowski, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," IEEE Trans. Software Eng., vol. SE-12, no. 5, pp. 662-675, May 1986.

[7] L. M. Ni, C. Xu, and T. B. Gendreau, "A Distributed Drafting Algorithm for Load Balancing," IEEE Trans. Software Eng., vol.SE-11, no. 10, pp. 1153-1161, October 1985.

[8] D. L. Eager and E. D. Lazowski, and J. Zahorjan, "A Comparision of Receiver-Inititated and Sender Initiated Adaptive Load Sharing," Performance Evaluation, 6, pp. 53-68, March, 1986.

[9] G. Cybenko, “Dynamic load balancing for distributed memory multiprocessors,” J. Parallel Distrib. Comput. 7 , pp. 279–301, 1989.

[10] J. Watts, S. Taylor, “A practical approach to dynamic load balancing,” IEEE Trans. Parallel Distrib. Systems 9 (3), pp. 235–248, March 1998.

[11] P. Krueger, N.G. Shivaratri, “Adaptive location policies for global scheduling,” IEEE Trans. Software Eng. 20 (6), pp. 432-444, June 1994.

[12] S. Dhakal, M. M. Hayat, J.E.Pezoa, C. Yang, and D. Bader, "Dyanmic Load Balancing in Distributed System in the Presence of Delays: A Regeneration-Therory Approach," IEEE Transactions on Parallel and Distributed Systems, vol. 18, no. 4, April 2007.

[13] D. J. Evans and W.U.N. Butt,” Dynamic load balancing using task-transfer probabilities,” Parallel Computing, Vol. 19, No. 8, pp. 897-916, August 1993.

[14] A. M. Alakeel, “Load Balancing in Distributed Computer Systems,” Int. Journal of Computer Science and Information Security, Vol. 8, No. 4, pp. 8-13, July 2010.

[15] A. M. Alakeel, “A Guide to Load Balancing in Distributed Computer Systems,” Int. Journal of Computer Science and Network Security, Vol. 10, No. 6, pp. 153-160, June 2010.

[16] K. Abini, “Fuzzy Decision Making for Load Balancing in a Distributed System, “ Proceedings of the 36th Midwest Symposium Circuits and Systems, pp. 500–502, 1993.

[17] Yu-Kwong Kwok, Lap-Sun Cheung, “A new fuzzy-decision based load balancing system for distributed object computing,” Journal of Parallel and Distributed Computing, Volume 64, Issue 2, pp. 238-253, February 2004.

[18] L. Singh, A. Narayan, and S. Kumar, "Dynamic fuzzy load balancing on LAM/MPI clusters with applications in parallel master-slave implementations of an evolutionary neuro-fuzzy learning system," IEEE International Conference on Fuzzy Systems, pp.1782-1788, June 2008.

[19] C.W. Cheong, V. Ramachandran, “Genetic Based Web Cluster Dynamic Load Balancing in Fuzzy Environment,” Proceedings of the Fourth International Conference on High Performance Computing in the Asia-Pacific Region, Beijing, China, Vol. 2, pp. 714–719, 2000.

[20] P. Chulhye, J.G. Kuhl, “A fuzzy-based distributed load balancing algorithm for large distributed systems,” Proceedings of the Second International Symposium on Autonomous Decentralized Systems, pp. 266–273, April 1995.

[21] M. Rantonen, T. Frantti, and K. Leiviskä, "Fuzzy expert system for load balancing in symmetric multiprocessor systems," Journal of Expert Systems with Applications, Vol. 37 No. 12, pp. 8711-8720, December, 2010.

[22] L. Singh, A. Narayan, and S. Kumar, "Dynamic fuzzy load balancing on LAM/MPI clusters with applications in parallel master-slave implementations of an evolutionary neuro-fuzzy learning system," IEEE International Conference on Fuzzy Systems, pp.1782-1788, June 2008.

[23] I. Barazandeh, S. S. Mortazavi, and A. M. Rahmani, "Intelligent fuzzy based biasing load balancing algorithm in distributed systems," IEEE 9th Malaysia International Conference, pp.713-718, Dec. 2009.

[24] E. El-Abd, “Load blancing in distrubuted computing systesms using fuzzy sexpert ssytems, “ Int. Confernce on Modern Probmesn of Radio Engiennering, Telecommunications and Computer Scinece, Lviv-Slavsko, Ukraine, pp. 141-144, 2000.

[25] S. Dierkes, “Load balancing with a fuzzy-decision algorithm,” Inform. Sci. 97 (1–2), pp. 159–177, March 1997.

[26] S. P. McAuliffe, “Job scheduling using fuzzy load balancing in distributed system,” Electron. Lett. 34 (20), pp. 1983–1985, October 1998.

[27] K.-W.Wong, “Fuzzy routing control of service request messages in an individual computing environment,” Proceedings of ACM Symposium on Applied Computing, Nashville, TN, pp. 548–551, 1995.

[28] A. Kumar, M. Singhal, and Ming T(M&e) Liu, "A Model for Distributed Decision Making: An Expert System for Load Balancing in Distributed Systems", IEEE computer software and applications conference, 1987.

[29] L. A. Zadeh, “Fuzzy Sets,” Information and Control, No. 8, pp. 338-353, 1965.

[30] B. Kosko, “Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence,” Prentice-Hall, Englewood Cliffs, NJ, 1992.

[31] J. Giarratano, “Expert Systems: Principles and Programming,” PWS-KENT Publishing Company, Boston, 1989.

[32] A. M. Alakeel, "A Fuzzy Dynamic Load Balancing Algorithm for Homogenous Distributed Systems,” Proceedings of the Int. Conference on Computer and Information Technology, Zurich, Switzerland, pp. 63-66, January 2012.

[33] S. Chowdhury, "The Greedy Load Sharing Algorithms," J. Parallel and Distributed Comput, vol. 9, pp. 93-99, May 1990.