Journal of Computer and Communications, 2024, 12, 110-122
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

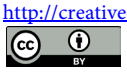
A Sender-Initiated Fuzzy Logic Contrnol Method for Network Load Balancing

Ming-Chang Huang

Department of Business Information System/Management Operations, University of North Carolina, Charlotte, USA
Email: mhuang5@charlotte.edu

How to cite this paper: Huang, M.-C. (2024) A Sender-Initiated Fuzzy Logic Contrnol Method for Network Load Balancing. *Journal of Computer and Communications*, **12**, 110-122.
https://doi.org/10.4236/jcc.2024.128007

Received: June 14, 2024
Accepted: August 23, 2024
Published: August 26, 2024



Abstract

In this paper, a sender-initiated protocol is applied which uses fuzzy logic control method to improve computer networks performance by balancing loads among computers. This new model devises sender-initiated protocol for load transfer for load balancing. Groups are formed and every group has a node called a designated representative (DR). During load transferring processes, loads are transferred using the DR in each group to achieve load balancing purposes. The simulation results show that the performance of the protocol proposed is better than the compared conventional method. This protocol is more stable than the method without using the fuzzy logic control.

Keywords

Load Balancing, Fuzzy Logic Control, Sender-Initiated

1. Introduction

Computer networks can provide computation parallelism. The imbalance in workloads among computers reduces the performance of the systems. To take advantage of the full capacity of these computer systems, load balancing and load sharing algorithms have been devised to improve the performance of the systems by the appropriate transfer of tasks among available computers.

As the configurations of today's computer networks are becoming more complicated, achieving high performance is becoming more challenging. There are several factors affecting the performance of computer networks. Among them, load balancing provides a helpful solution to balance or share workloads among computer systems. Distributed algorithms usually use local information for transferring excessive loads in heavily loaded nodes to lightly loaded nodes. In conventional load balancing methods, fixed threshold levels are used to decide

whether a node is heavily or lightly loaded [1]. This approach leaves room for improving the performance of distributed systems by applying fuzzy logic control to the threshold levels.

In general, two methods are used – static and dynamic. 1) Static method –in this method, threshold levels are fixed and are not changed according to the current status of the system. Therefore, nodes in the system do not exchange state information for choosing new threshold level (s). 2) Dynamic method—in this method, threshold levels are changed according to the current status of the system. State information exchange is necessary when this method is used. Thus, dynamic methods react to changes in the system state compared to static methods. In general, dynamic load-balancing algorithms can respond better to system changes and result in better performance. There have been several studies of dynamic load transfer policies [1]-[10].

The use of a fixed threshold value may lead to fruitless load transfers and make the scheduling algorithm unstable because a node's load may be below the threshold when it decides to accept a remote process, but its load may become larger than the threshold level as soon as the remote process arrives [11]. Therefore, immediately after receiving the remote load, the node will again try to transfer one or more of its processes to some other nodes.

Alonso and Cova [12] proposed a double-threshold level algorithm "high-low policy" to reduce the instability of the single-threshold level policy. The high-low policy uses two threshold values called high mark and low mark and therefore divides the space into three regions—overloaded, normal, and underloaded regions. However, those two load threshold levels are fixed and predefined. One desired feature of a modern distributed system is to change the status of nodes dynamically. Furthermore, excessive traffic and congestion reduce the performance of computer networks. Some of the load-transferring algorithms and network congestion control algorithms use pre-defined threshold levels [11]-[15].

Fuzzy logic control is a natural extension to the two-threshold level approach, and it improves the performance without pre-defining the threshold values. The fuzzy logic approach can deal with a system's changes for balancing loads and improve system's performance more efficiently.

Chulhye and Kuhl [16] used fuzzy logic control for load balancing in distributed systems. However, they assume that every node knows the load status of every other node in its group by exchanging load information periodically. This generates too much traffic in the computer networks. In this paper, a new protocol is presented for load balancing, which uses fuzzy logic control and multicast method and is suited for computer networks. This approach finds available nodes by employing a hierarchical lookup service.

The system model is discussed in Section 2. Section 3 includes the details of the protocol. The simulation results are shown in Section 4 and Section 5 presents the conclusion.

2. Fuzzy Logic Control

Dr. Zadeh proposed fuzzy logic concept in 1965 [13]. Fuzzy logic is a powerful tool in representing linguistic information and is especially useful to solve problems that do not have a precise solution and the conventional methods cannot solve them very well. For example, a computer load can be regarded as a linguistic variable and its value can be considered as—light, moderate, and heavy. Figure 1 shows an example, where if the load is between 0 and 30, or between 10 and 70, or more than 50, then the computer is considered as lightly, moderately, or heavily loaded, respectively. In this figure, the horizontal axis represents a computer load, and the vertical axis shows the membership function and the degree a computer load is lightly, moderately, or heavily loaded. For example, point A in this figure represents that the computer load is 100% lightly loaded, in contrast to point B where the computer load is considered 80% lightly loaded and 40% moderately loaded.

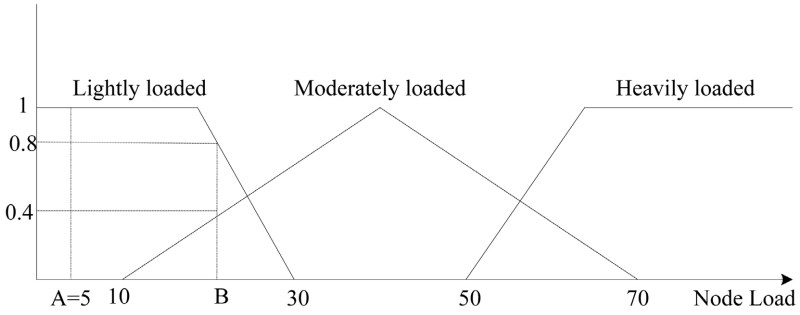


Figure 1. Fuzzy functions for the load of nodes.

In the conventional load balancing schemes with a single threshold level, a computer system load status is considered either lightly loaded or heavily loaded depending on whether its load is below or above a threshold level. Therefore, a computer system, which is lightly loaded, may suddenly become heavily loaded by receiving an extra load. In contrast, in a fuzzy logic control, a computer system load status gradually changes from the lightly loaded status to the heavily loaded status depending on its membership function, which represents a more realistic change of load status than conventional methods. Also, a system has more potential to be stable than conventional methods due to reducing unnecessary jobs movements and reducing thrashing probability.

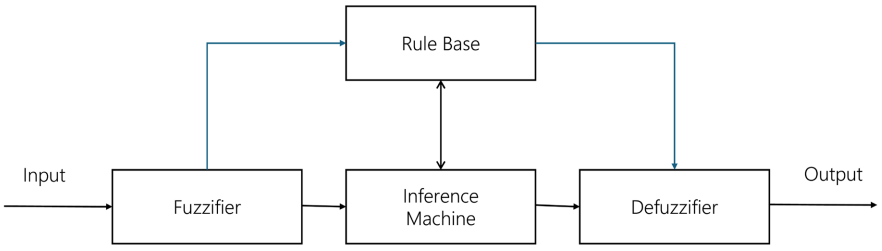


Figure 2. Fuzzy logic controller architecture.

Usually, a fuzzy logic control scheme includes inputs, an inference mechanism, and an output. After the input variables are determined, a rule-based engine can be set in the form of "IF A and B THEN C," where A, B, and C are fuzzy sets. Figure 2 shows the fuzzy logic controller structure that consists of fuzzifier, inference machine, defuzzifier, and rule base components.

Fuzzifier converts the input values to degrees of membership via the membership functions. Usually, the degrees are between 0 and 1. The inference machine implements rule references according to the fuzzy rule base in order to generate outputs. Inference rules use the MAX-MIN method to generate the output degrees. Defuzzifier then uses the output degrees to calculate the output crisp value. Several methods are used to find the crisp value by the defuzzifier [13]. The most commonly used method is the centroid method. This procedure is the most prevalent and physically appealing of all the defuzzification methods.

3. System Model

In this study, it is assumed that underlying computer networks have a hierarchical structure. For example, as shown in Figure 3, several nodes form a group and each group contains a node called designated representative (DR), which communicates with the DRs in other peer groups. The DR node for a group is selected to be the node with the lowest workload at the beginning in that group. Group 1 contains three nodes—A.1.1, A.1.2, and A.1.3 and A.1.2 is the DR in this group. Group 2 contains four nodes – A.2.1, A.2.2, A.2.3, and A.2.4 and A.2.2 is the DR of the group. Group 3 contains four nodes – B.1, B.2, B.3 and B.4 and B.1 is the DR of the group. All these three groups are physical groups. Group 4 is a logical group at the next level of the hierarchy. Each node in a logical group is a logical node that represents the whole lower group for the next level in the hierarchy. For example, logical node A.1 represents Group 1 and logical node A.2 represents Group 2.

This hierarchical structure is similar to the computer network structure where each of the physical groups 1, 2, and 3 may represent a Local Area Network (LAN), and each of the next level groups, like Group 4, may represent a Metropolitan Area Network (MAN), and each of the higher groups, like Group 5, may represent a Wide Area Network (WAN). This structure is similar to Internet and Asynchronous Transfer Mode (ATM) network structures. In this model, it is assumed that logical nodes at upper levels are represented by the physical nodes at the lowest levels. For example, physical nodes A.1.2 and A.2.2 represent logical nodes A.1 and A.2, respectively.

4. Fuzzy-Logic Load Balancing Approach

The approach is composed of the following main steps.

4.1. Group Forming

In order to reduce the communication overhead, nodes at close distance may form a group. For instance, all or some of the nodes in the same Local Area

Network (LAN) may form a group. In the proposed system model, nodes may join or leave their groups dynamically. A node may join a group by sending a message to its group's DR. For a node to leave a group, the node has to send its entire load to some other nodes. This step is implemented as follows:

- The leaving node sends a "LEAVE" message to its DR about its leaving decision.
- Then the DR finds suitable nodes for load transfer.
- The leaving node will drop its loads in its queue. If its loads are not completely transferred after some time period, then the remaining jobs in the queue are dropped. The time period can be variable and set to, for instance, at least twice the maximum allowance of waiting time for making a load transfer request.

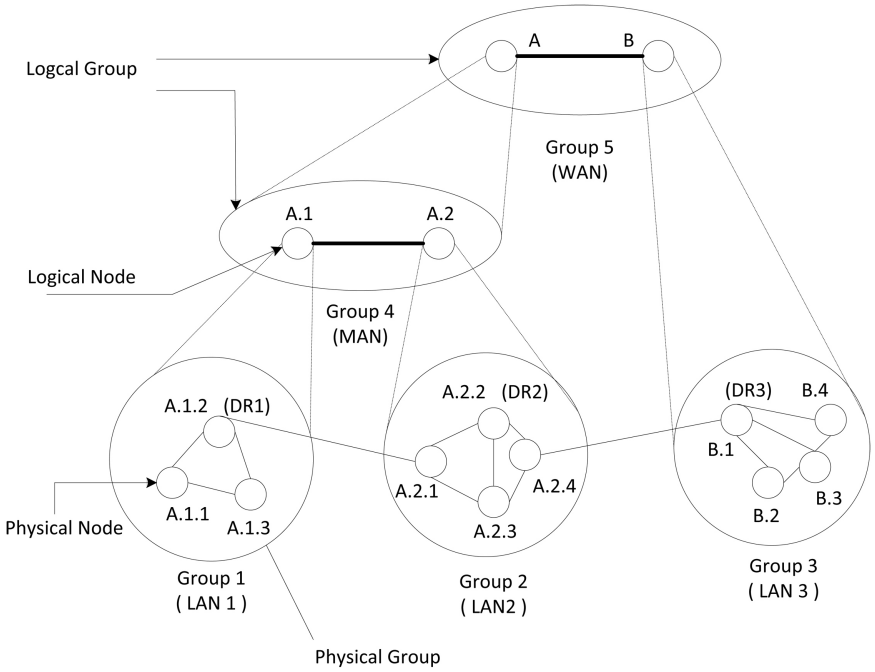


Figure 3. System configuration for load balancing.

4.2. Fuzzy Logic Load Balancing Algorithm

The algorithm for fuzzy-based load balancing has the following steps.

- *Fuzzification*

Every node should evaluate its own load status. First, the workload of a node is determined. Then it is used as the input to its membership functions for finding its degree according to Table 1, which is explained later.

- *Applying the inference rules*

A heavily loaded node may send a request for finding a lightly loaded node for the load transfer using the fuzzy logic inference rules.

- *Defuzzification*

Centroid method is applied to determine the output crisp value (the load transfer probability). Once the workloads for the sender and receiver are known

and the inference rule is applied, then the output crisp value is found from the output membership functions.

- *Find the suitable node for the load transfer*

The easiest way for a requesting node to find a suitable node is to choose the lightly loaded node that is the first one to reply. There might be several nodes that are lightly loaded for load transfer and the requesting node may choose the least lightly loaded node. One method is to choose the node that has the highest crisp value after the defuzzification process.

- *Transfer the loads from the sender to the receiver*

It is possible that some of the senders may request the same receiver at the same time. The system performance may degrade if the receiver accepts all the requests. To prevent this to happen, the receiver may choose only the first request and reject the others.

4.3. Fuzzy Logic Control

To apply fuzzy logic control for exchange of loads between two nodes, in the Fuzzification step, the input variables have to be determined first. Two input variables are needed – the current load of the sender and the current load of the receiver. The output is the probability for the load transfer. The bigger the output is, the higher the probability of transferring the loads of the node. Five fuzzy subsets are defined for the input variables – VL, L, M, H, and VH, which represent very lightly loaded, lightly loaded, moderately loaded, heavily loaded, and very heavily loaded states, respectively. Similarly, five subsets are defined for the output variable – VL, L, M, H, and VH, which similarly represent the very-low, low, medium, high, and very-high probabilities for load transfer between the two nodes. The membership functions for input and output variables are shown in Figure 4 and Figure 5 separately, and the rules for the rule base are shown in Table 1.

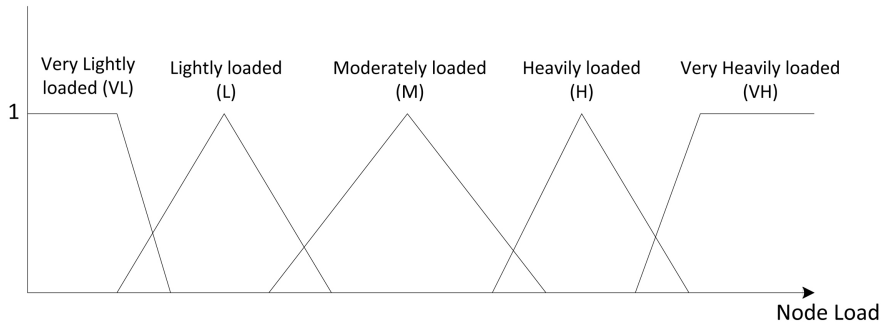


Figure 4. Membership functions for input variables.

4.4. Sender-Initiated Protocol

An end node is a leaf node in the hierarchical structure of the network. Each node checks its load status when a job arrives, which applies Fuzzy Logic Control (FLC) to decide whether a load transfer is necessary. In a homogeneous sys-

tem, every node has the same membership function, and each node can use the same membership function. However, nodes may have different membership functions in a heterogeneous system. Therefore, the receiver has to send its membership function parameters to the load requester applying the fuzzy logic control to determine the probability of the load transfer.

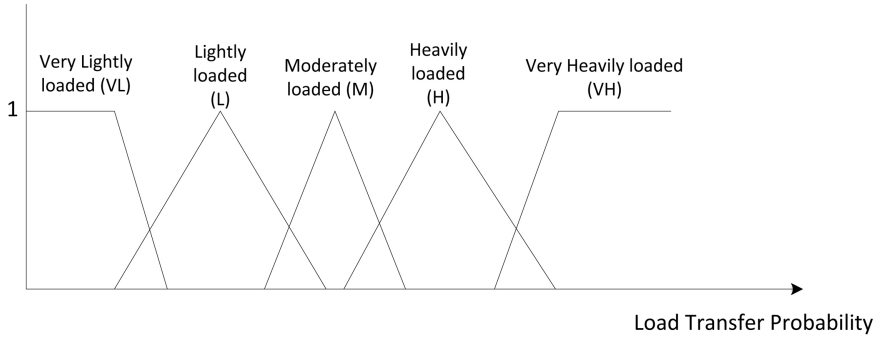


Figure 5. Membership functions for output variables.

Table 1. Inference rules for fuzzy logic control in load balancing.

Receiver Load (*RL*)	VL	L	M	H	VH
VL	VL	L	M	H	VH
L	VL	VL	L	M	H
M	VL	VL	VL	L	M
H	VL	VL	VL	VL	L
VH	VL	VL	VL	VL	VL

For the sender-initiated protocol, an end node sends a load transfer request to other nodes when it is heavily loaded. First, the way a heavily loaded end node finds another node for load transfer is explained. At the beginning of this process, a heavily loaded end node uses multicast method to send a request to other nodes in its own group asking for load transfer. If there is no lightly loaded node available in its group, then the node sends a request to the DR of its group for finding a lightly loaded node in other groups. Next, the DR of its group arbitrarily chooses a DR in the same logical group and sends a load transfer request to that DR. After receiving that request, the DR multicasts a load transfer request to the nodes in its group. The details of the algorithms are discussed below.

4.4.1. Algorithm for an End-Node Using Sender-Initiated Protocol

When an End-Node is heavily loaded, it will do:

1) Multicast a "LOAD-TRANSFER" message to all the nodes within its own group.

2) Wait to receive replies from the nodes within the group which are not heavily or very heavily loaded. The wait time could be equal to the maximum

delay in the network.

3) If some replies are received, then choose the least lightly loaded node among them and run the fuzzy logic rule based control to get the output crisp value P—the probability for the load transfer. Then transfer M loads to the selected node where $M = P \times (LS - LR)/2$, LS is the load of the sender and LR is the load of the receiver.

4) Otherwise, send a "LOAD-TRANSFER" message to the DR of its group. The DR will send this message out to other DRs for finding some other lightly loaded end nodes in other groups for the load transfer.

a) If received replies from some other nodes in other groups, choose the least lightly loaded node among them and run the fuzzy logic rule based control to get the output crisp value. Then, transfer M loads to the selected node where $M = P \times (LS - LR)/2$. And send a "FOUND" message to the DR.

4.4.2. Algorithm for a DR Node Using Sender-Initiated Protocol

When a DR node receives a "LOAD-TRANSFER" message, it will do:

1) When a DR node receives the "LOAD-TRANSFER" message from an End-Node within its own group, then the DR node sends a "LOAD-TRANSFER" message to another DR node in its logical group.

2) When a DR node receives a "LOAD-TRANSFER" from another DR, then it multicasts a "LOAD-TRANSFER" message to the End-Nodes within its group.

3) The DR node, which has sent a "LOAD-TRANSFER" message, waits for a time out equal to the maximum delay in the network.

a) If the DR node receives a "FOUND" message, then it terminates the search and relays the "FOUND" message to the last DR that sent a "LOAD-TRANSFER" message.

b) If the DR does not receive a "FOUND" message within a time out equal to the maximum delay in the network, then it should poll another DR within its logical group if one exists and send a "LOAD-TRANSFER" message to it or else send a "LOAD-TRANSFER" message to its own DR in the next upper level in the network hierarchical tree structure.

c) The DR node should continue the search until either it successfully finds a lightly loaded node or exhaustively searches the entire network hierarchical tree structure.

Example: Assume that all the nodes in Group 1 (LAN 1) in **Figure 3** are heavily loaded. Thus, a heavily loaded node A.1.1 has to find another node in other groups for load transfer. First node A.1.1 sends a request to its DR (node A.1.2) for load transfer. Logical node A.1 in Group 4 represents the nodes in Group 1. Logical node A.1 (acting as the DR in Group 1) sends a request to the logical node A.2 in Group 4 for finding a lightly loaded node in Group 2. Logical node A.2 represents the nodes in Group 2 (LAN 2) and acts as the DR of Group 2. Therefore, A.2.2 (the DR of Group 2) uses multicast to send requests to all the nodes within Group 2. If there are some nodes lightly loaded in Group 2, then they will reply directly to the original node A.1.1 that sent the load transfer re-

quest. At this time, the original node A.1.1 selects the least lightly loaded node and starts transferring loads directly to that node and sends a "FOUND" message to its DR to stop the search.

5. Preliminary Simulation Results and Analysis

5.1. Simulation Environment

Network simulator NS-2 [17] is used for the simulation study. Figure 6 shows the system used for the simulation. There are three groups in the system with 3 nodes in each group. Each group represents a local area network. UDP is used in the system. The communication speed for link is 5 Mb/sec. Assume the mean job arrival rate for each node is λ and the service rate for each node is S, then the utilization for that node $U = \lambda S$. It is assumed that S is equal to 50 time units and the utilization for all the nodes in the system is less than 1 except for Node 1 in Group 1, which generates jobs more than that its queue can store. This forces the node to transfer some of its load to other nodes when its load is above the heavily loaded threshold level. Of course, some arriving jobs are dropped when its queue is full. The membership functions change is included in the results.

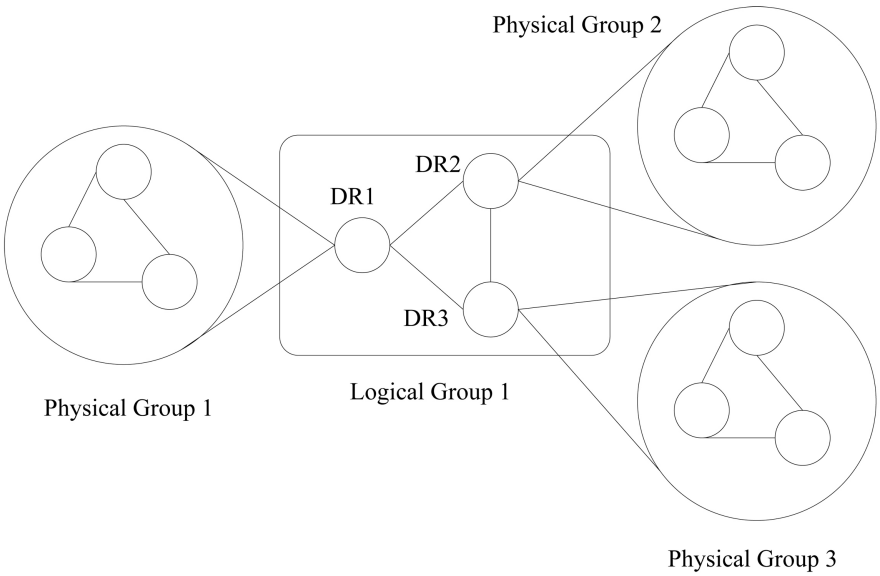


Figure 6. System model for simulation.

5.2. Algorithms for Comparison

The performance and job drop rate are compared for three algorithms—BID [18], Non-Fuzzy and Fuzzy algorithms.

5.2.1. BID Algorithm

A node multicasts a request to the other nodes in the same group when it is heavily loaded. The other node sends an acknowledge back to the sender when it receives the request, and it is lightly loaded. If none of the nodes in the group are available for load transfer, the sender waits for a period of time and then

re-sends the request again instead of sending its request to the nodes in other groups.

5.2.2. Non-Fuzzy Load Transfer Algorithm

Unlike the BID algorithm that can only send requests to the nodes in the same group, the Non-Fuzzy load transfer algorithm may send requests to nodes in other groups to ask for load transfer. However, it uses the fixed threshold levels for load transfer without applying fuzzy logic control in it.

5.3. Performance Comparison for Sender-Initiated Protocol

The performance of a node is measured by the execution time of jobs in that node. Figure 7 and Figure 8 show the results of performance comparison and drop rate comparisons when 5000 jobs are executed. The system performance and drop rates for Fuzzy and Non-Fuzzy algorithms are found to be better than those of BID.

There is no significant difference of system performance between Fuzzy and Non-Fuzzy algorithms. However, the drop rate of Fuzzy algorithm is better than that of Non-Fuzzy algorithm when the node utilization is higher than 0.8. This suggests that the load transfer with fuzzy algorithm is smoother and more stable than the Non-Fuzzy algorithm.

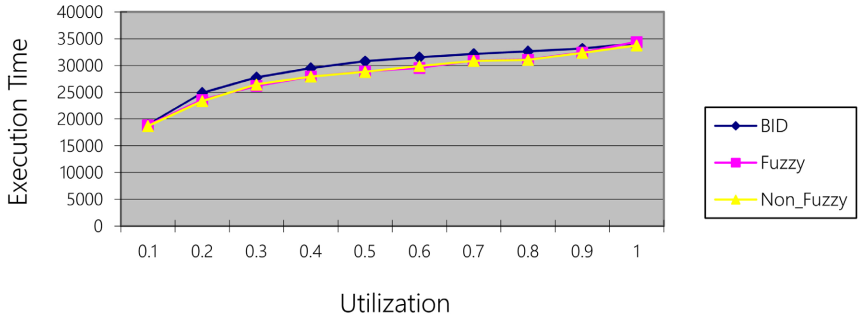


Figure 7. Performance comparison with one busy node.

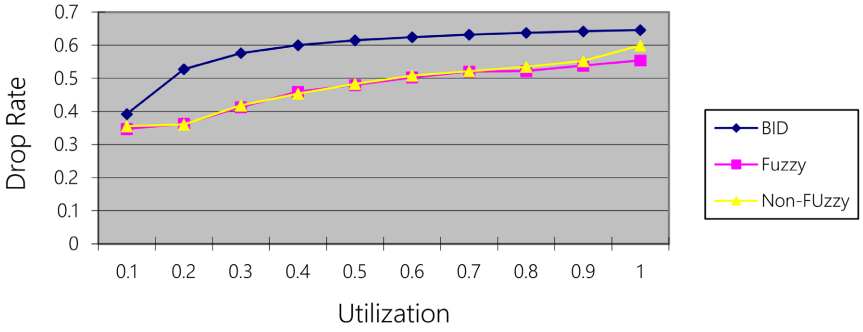


Figure 8. Drop rate comparison with one busy node.

Two nodes are now set in Group 1 with their arrival rate greater than the service rate. Therefore, it is hard for those two nodes to find available nodes in

Group 1 to share loads with. This causes the heavily loaded nodes to transfer loads to other nodes in another group via network communication. Figure 9 and Figure 10 show the results of the simulation. The system performance using Fuzzy and Non-Fuzzy algorithms is again better than the system performance using BID algorithm. The drop rate for Fuzzy algorithm is also lower than that for BID and Non-Fuzzy algorithm. This indicates that the fuzzy load transfer algorithm can lead to better performance than the other two.

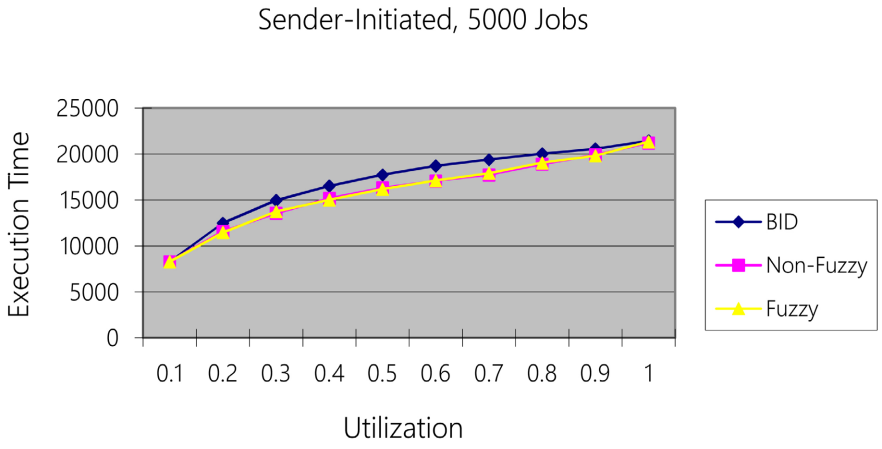


Figure 9. Performance comparison with two busy nodes.

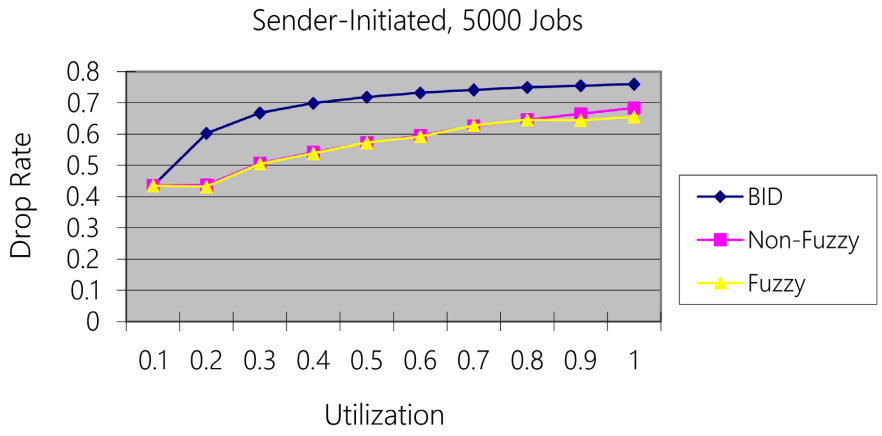


Figure 10. Drop rate comparison with two busy nodes.

6. Conclusions

In this paper, a new protocol is introduced for load balancing using fuzzy logic control, multicast, and active network concepts. This new protocol is based on a logical hierarchical structure that locates nodes for load transfer dynamically. Multicast method is applied in order to save the bandwidth of the network. Nodes in the computer networks can adjust their membership functions according to their load status and traffic congestion status. The simulation results show that the performance of the protocol is better than that by using BID method for the system structure and data used in the study.

An interesting feature of the protocol is that it first tries to find nodes at close distance for the load transfer over remote nodes. It has shown from the simulation results that the protocol is stable and smooth because no fixed threshold levels are used and there is a smooth transition from lightly or heavily loaded status to moderate status.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Eager, D.L., Lazowska, E.D. and Zahorjan, J. (1986) Adaptive Load Sharing in Homogeneous Distributed Systems. *IEEE Transactions on Software Engineering*, **12**, 662-675. https://doi.org/10.1109/tse.1986.6312961

[2] Hosseini, S.H. (2000) Special Issue on Load Balancing. *Cluster Computing Journal*, **3**.

[3] Hosseini, S.H., Litow, B., Malkawi, M. and Vairavan, K. (1987) Distributed Algorithms for Load Balancing in Very Large Homogeneous Systems. *Proceedings of ACM-IEEE Fall Joint Computer Conference*, **29**, 397-404.

[4] Hosseini, S.H., Litow, B., Malkawi, M., McPherson, J. and Vairavan, K. (1990) Analysis of a Graph Coloring Based Distributed Load Balancing Algorithm. *Journal of Parallel and Distributed Computing*, **10**, 160-166. https://doi.org/10.1016/0743-7315(90)90025-k

[5] Kremien, O. and Kramer, J. (1992) Methodical Analysis of Adaptive Load Sharing Algorithms. *IEEE Transactions on Parallel and Distributed Systems*, **3**, 747-760. https://doi.org/10.1109/71.180629

[6] Kumar, A. (1989) Adaptive Load Control of the Central Processor in a Distributed System with a Star Topology. *IEEE Transactions on Computers*, **38**, 1502-1512. https://doi.org/10.1109/12.42120

[7] Lin, F.C.H. and Keller, R.M. (1987) The Gradient Model Load Balancing Method. *IEEE Transactions on Software Engineering*, **13**, 32-38. https://doi.org/10.1109/tse.1987.232563

[8] Mirchandaney, R., Towsley, D. and Stankovic, J.A. (1989) Adaptive Load Sharing in Heterogeneous Systems. *The 9th International Conference on Distributed Computing Systems*, Newport Beach, 5-9 June 1989, 298-306.

[9] Mirchandaney, R., Towsley, D. and Stankovic, J.A. (1989) Analysis of the Effects of Delays on Load Sharing. *IEEE Transactions on Computers*, **38**, 1513-1525. https://doi.org/10.1109/12.42124

[10] Wolffe, G.S. (1998) Scheduling and Load Balancing for Distributed Systems. Ph.D. Thesis, University of Wisconsin.

[11] Sinha, P.K. (1996) Distributed Operating Systems: Concepts and Design. IEEE Press.

[12] Alonso, R. and Cova, L.L. (1988) Sharing Jobs among Independently Owned Processors. *Proceedings of the 8th International Conference on Distributed Computing Systems*, San Jose, 13-17 June 1988, 282-288.

[13] Floyd, S. and Jacobson, V. (1993) Random Early Detection Gateways for Congestion

Avoidance. *IEEE/ACM Transactions on Networking*, **1**, 397-413. https://doi.org/10.1109/90.251892

[14] Jacobson, V. (1988) Congestion Avoidance and Control. *ACM SIGCOMM Computer Communication Review*, **18**, 314-329. https://doi.org/10.1145/52325.52356

[15] Ramakrishnan, K.K. and Jain, R. (1990) A Binary Feedback Scheme for Congestion Avoidance in Computer Networks. *ACM Transactions on Computer Systems*, **8**, 158-181. https://doi.org/10.1145/78952.78955

[16] Chulhye, P. and Kuhl, J.G. (1995) A Fuzzy-Based Distributed Load Balancing Algorithm for Large Distributed Systems. *Proceedings ISADS* 95. *Second International Symposium on Autonomous Decentralized Systems*, Phoenix, 25-27 April 1995, 266-273.

[17] UCB/LBNL/VINT Network Simulator. http://www.isi.edu/nsnam/ns/index.html

[18] Smith, R. (1980) The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, **29**, 1104-1113. https://doi.org/10.1109/tc.1980.1675516