A Fuzzy Load Balancing Service for Network Computing Based on Jini†

Lap-Sun Cheung and Yu-Kwong Kwok

Department of Electrical and Electronic Engineering
The University of Hong Kong, Pokfulam Road, Hong Kong
Email: *{lscheung, ykwok}@eee.hku.hk*

Abstract. Distributed object computing systems are widely envisioned to be the desired distributed software development paradigm due to the higher modularity and the capability of handling machine and operating system heterogeneity. As the system scales up (e.g., with larger number of server and client objects, and more machines), a judicious load balancing system is required to efficiently distributed the workload (e.g., the queries, messages/objects passing) among the different servers in the system. However, in existing distributed object middleware systems, such a load balancing facility is lacking. In this paper, we describe the design and implementation of a dynamic fuzzy-decision based load balancing system incorporated in a distributed object computing environment. The proposed approach works by using a fuzzy logic controller which informs a client object to use the most appropriate service such that load balancing among servers is achieved. We have chosen Jini to build our experimental middleware platform, on which our proposed approach as well as other approaches are implemented and compared. Extensive experiments are conducted to investigate the effectiveness of our fuzzy-decision based approach, which is found to be consistently better than other approaches.
Keywords: distributed object computing, load balancing, fuzzy decision, Java, Jini, remote method invocation, middleware.

1 Introduction

With the great advancement of hardware technologies, powerful distributed computing systems are becoming ubiquitous. Indeed, with commodity hardware components, a high performance network of PCs can be set up to execute applications developed using new software structuring paradigms, such as object based systems and object brokerage protocols, which have also advanced tremendously parallel to the development of hardware technologies. Such new distributed software development paradigms, while have the advantages of modularity and capable of handling platform heterogeneity, were conceived as impractical in mere five to ten years ago because many complex operations such as object serialization and data marshalling, were too time consuming to be efficiently run on the hardware platforms. Currently, many commercial software projects are using distributed object based approaches such as CORBA (common object request broker architecture), DCOM, and Java RMI (remote method invocation). Using a distributed object based approach, an application is constructed as a group of interacting objects. These objects are distributed over multiple machines which interact with each other through well predefined protocols (e.g., RMI in Java). Usually, the interactions are queries or remote services invocation.

It is common that in such a distributed object computing system, there are multiple objects (possibly on heterogeneous platforms) that provide the same service. This is done to achieve a higher availability and scalability. Even the lookup service object may have several instances in the network. Under light load conditions (i.e., few number of remote service invocations or

†. This research was jointly supported by the Hong Kong Research Grants Council under contract numbers HKU 7124/99E and HKU 7024/00E), and by a HKU URC research grant under contract number 10203413.

R. Sakellariou et al. (Eds.): Euro-Par 2001, LNCS 2150, pp. 183-190, 2001.

object passing), the system can perform reasonably well. However, as the system scale up to a moderate size (e.g., 10 machines), the number of requests generated in the system can be of a very large volume and be very bursty. As a result, some machines might be overloaded while other machines are idle or lightly loaded. In order to improve the performance, specifically the client request response time of a distributed application, load balancing techniques can be introduced to distribute workload in a judicious manner among various machines [14].

We focus on the middleware based approach to perform load balancing in a distributed object network. The major requirements of a middleware based load balancing service in a object computing system are:

- *Client transparency*: Client programs need not be modified in order to use load balancing service. Client should not be aware of the changes due to incorporating of a new load balancing algorithm in the load balancing service.
- *Avoid changing middleware layer*: Middleware layer should not be modified. Once the middleware layer has been modified, it will become proprietary and incompatible with other existing applications developed using the original middleware.
- *Server transparency*: Like client transparency, server applications need not be changed so as to take the advantage of load balancing service.
- *Scalability and fault-tolerance*: To avoid a single point of failure, several load balancers can coexist in a federation. Load balancers can be configured to work cooperatively to form a single logical load balancing service.
- *Integrating new load balancing algorithms*: The load balancing service should be designed in such a way that different load balancing algorithms can be easily integrated without extensive modification of the source code of the load balancing service, i.e., modular design pattern should be employed to develop the load balancing service.
- *Minimal overhead*: Network overhead caused by load balancing service should be minimized. A load balancing service should be designed such that unnecessary message/ object exchanges between the load balancing service with other network components should be avoided; otherwise, the overall system performance cannot be guaranteed.

Existing load balancing algorithms used in distributed object computing systems are usually based on simple techniques such as round-robin or random, which may not give optimized performance. Thus, in our study we propose, implement, and evaluate a new approach for load balancing in a distributed object system. We incorporate our load balancing scheme in a dynamic network service system called Jini [7]. Our proposed load balancing algorithm employs fuzzy-decision control [8]. An effective load balancing scheme requires the knowledge of the global system state (e.g., the workload distribution). However, in a distributed computing system, the global state is swiftly and dynamically changing and it is very difficult to accurately model the system analytically [4], [6]. Thus, in order to tackle the load balancing problem in such an environment where state uncertainty is unavoidable, we employ a fuzzy-decision approach to model those state variables that cause uncertainty in global states. Our approach is novel in that it is seamlessly incorporated into the Jini middleware. Our fuzzy-decision control is also effective and robust, as evident in the experimental results in a real Jini network, due to the concise fuzzy rules set. There have been some recent attempts in using fuzzy-based approaches for load balancing [2], [3], [5], [13], but those approaches are either too restrictive or not suitable for a middleware based environment considered in our study.

This paper is organized as follows. In the next section, we describe our proposed fuzzy-decision based approach. Section 3 contains the experimental results we obtained in our Jini-based testbed built on a Pentium PCs network. The last section concludes this paper.

2 The Proposed Fuzzy-Decision Based Load Balancing Scheme

In this section, the design of the our proposed fuzzy-decision based load balancing service is described. We first describe the fuzzy logic controller, which is the core part of the fuzzy load balancing service. The dynamic interactions between the fuzzy load balancing service and other

components are discussed in detailed in subsequent subsections.

To handle a complex system such as a high speed computer network where a lot of uncertain parameters exist, a model with complex and nonlinear relationships between a lot of variables have to be devised, making a conventional control theory based approach intractable. To overcome this problem, fuzzy logic control theory [8] can be applied instead of the conventional one. Fuzzy logic control attempts to capture intuition in the form of `IF-THEN` rules, and conclusions are drawn from these rules [8]. Based on both intuitive and expert knowledge, system parameters can be modeled as linguistic variables and their corresponding membership functions can be designed. Thus, nonlinear system with great complexity and uncertainty can be effectively controlled based on fuzzy rules without dealing with complex, uncertain, and error-prone mathematical models [8]. The architecture of the fuzzy logic controller shown in Figure 1 includes five components: Fuzzifier, Rule Base, Membership functions, Fuzzy Inference Engine, and Defuzzifier. The fuzzifier is the input interface which maps a numeric input to a fuzzy set so that it can be matched with the premises of the fuzzy rules defined in the application-specific rule base. The rule base contains a set of fuzzy `if-then` rules which define the actions of the controller in terms of linguistic variables and membership functions of linguistic terms. The fuzzy inference engine applies the inference mechanism to the set of rules in the fuzzy rule base to produce a fuzzy set output. This involves matching the input fuzzy set with the premises of the rules, activation of the rules to deduce the conclusion of each rule that is fired, and combination of all activated conclusions using fuzzy set union to generate fuzzy set output. The defuzzifier is an output mapping which converts fuzzy set output to a crisp output. Base on the crisp output, the fuzzy logic controller can drive the system under control.

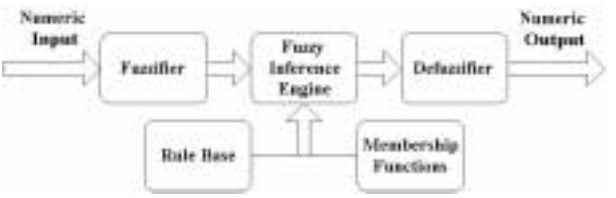


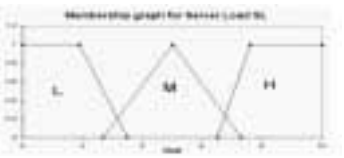
Figure 1: The architecture of the fuzzy logic controller.

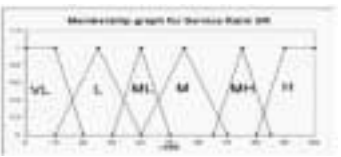
The fuzzy rule base contains a set of linguistic rules. These linguistic rules are expressed using linguistic values and linguistic variables. Different linguistic values can be assigned to a linguistic variable. For instance, `very_long` or `moderately_short` can be used in the variable `remote_method_Invocation_time`. These linguistic values are modeled as fuzzy sets. Based on the linguistic values, their corresponding membership functions can be expressed based on application requirements.

In the Jini computing model, client objects basically have no idea on which service they should send requests to in order to achieve the best QoS (quality of service) or more specifically, the shortest response time. Generally, a request router or load balancer can be implemented to route client requests to the most appropriate service. The request router can make such decisions based on the current state of server objects. However, such state information may not be updated and reliable as client requests reach servers [11]. That is the state information cannot reflect the state of servers accurately due to network delay. The request router needs to use approximate reasoning to handle the fuzzy information so as to make the system efficient. In order to make a correct routing decision, linguistic variables, server load, remote method invocation time, and service rank, are used in the fuzzy logic algorithm and are defined as follows.

We define server load, denoted as SL, with the fuzzy set definition: {`low (L), medium (M), high (H)`}. Accurate estimate of load is notoriously difficult to obtain [9], [10]. We employ an indirect approach in determining SL. Instead of directly measuring each process execution time, we measure the execution time of a benchmark program which consists of several benchmark kernel loops. The benchmark program runs perpetually without stopping in

the system as a background process. By observing the running times of the benchmark program, we can infer the instantaneous load level in the system. Figure 2 shows the membership graph for SL.





fuzzy inference rules

RMIT/SL	L	M	H
S	H	MH	ML
M	MH	M	L
L	M	L	VL

Figure 2: The membership graphs for server load (SL), RMI time (RMIT), service rank (SR), and the fuzzy inference rules.

In order to measure the responsiveness of servers to client requests as well as the overhead introduced into the network by distributed processing (i.e., message/object exchanges), network utilization needs to be determined. This is done by measuring the time for Remote Method Invocation. We define a benchmark remote method which simply returns a primitive data type from server to our fuzzy logic load balancing service to measure the RMI time:

```
int n; public int getNumber() {return n;}
```

The method `System.currentTimeMillis()` is used to measure the time elapsed during remote method invocation in milliseconds. In our measurements, it is found that the time needed to execute the remote method is about 2 to 3 ms when network utilization is low and server is lightly loaded. The time becomes longer when the system load increases. Thus the benchmark remote method can approximately reflect the network load and the responsiveness of servers. The fuzzy set of remote method invocation time (RMIT) is defined as: {`short (S), medium (M), long (L)`}. Figure 2 shows the membership graph for remote method invocation time RMIT.

We use service rank (SR) to classify services into six different categories. The fuzzy set of SR is: {`very low (VL), low (L), medium low (ML), medium (M), medium high (MH), high (H)`}. The membership graph of service rank is shown in Figure 2. The higher the rank that a service gets, the more appropriate that it can accept client requests. After defining the above fuzzy variables, a set of inference rules is defined as shown in Figure 2. By applying the fuzzy inference rules, a decision can be generated based on both antecedents. That is, if RMIT is short and SL is low, then SR is high. Having these fuzzy inference rules and membership graphs, the fuzzification and defuzzification processes can be carried out as follows. First, the input values of RMIT and SL are mapped to their respective membership degree values on their membership graphs. These degree values are compared and the minimum of the two is then projected onto the membership function of their consequence graph. The output graph, usually in the shape of a trapezium [8], then represents the output of one inference rule. After the output graph is generated, defuzzification of the fuzzy output into a crisp or numeric value can be carried out. We used the centroid method [8] to defuzzify the output. It has been noted that the thundering herd effect [12] may occur if the load balancer immediately forwards all requests to a server that is assigned to the highest service rank. It will lead to a sudden degradation in the overall system performance. In order to minimize such effect, the fuzzy logic load balancing service schedules client requests to servers based on a prioritized round-robin algorithm [1]. In our implementation using the Jini platform, the system consists of a lookup service, a fuzzy logic load balancing service, several server objects (providing services)

and client objects. In the following, we discuss the structure and the function of each component.

Each standard Jini service consists of basic functions such as finding lookup service and registering to it. These actions can be accomplished by installing a `JoinManager`, which is a standard Jini feature, into the service. Figure 3 shows the services residing in the server host. In our implementation, there is a general service in which clients are interested. The service consists of a `JoinManager`, `Remote Service Object` and `Remote Service Admin Object`. `Remote Service Object` is a remote object which implements the remote interface that is known to client. Client can invoke methods on service object based on the remote interface. `Remote Service Admin Object` provides an interface so that the service attributes can be modified by other administrative components in the network. We also incorporate a load monitor service in the server host. Like other Jini services, load monitor service registers itself with the lookup service with the help of `JoinManager`. The remote load monitor service object is being used when the fuzzy logic load balancing service registers as a listener for the remote event generated by load monitor service. The event consists of information such as server location and server load.

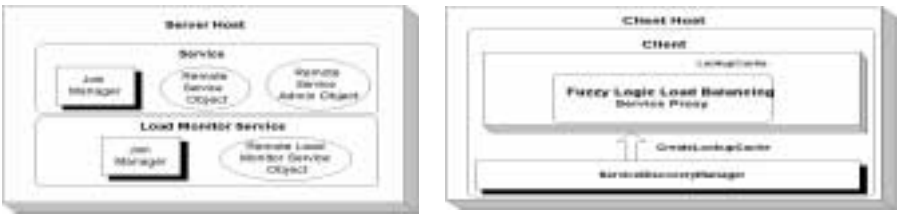


Figure 3: Structure of the server and client hosts.

Basically, a client locates lookup service and gets the services proxies available in the lookup service. Then the client, based on the service interface, generates a request to fuzzy logic load balancing service and gets the result after the service has finished the execution. The structure of the client is shown in Figure 3. The client consists of a `LookupCache` which is created by `ServiceDiscoveryManager`. Inside the cache, it stores the fuzzy logic load balancing service proxy to which client can send request. `ServiceDiscoveryManager` is a standard Jini feature of which the main use is to help clients to locate services and cache service proxies. Fuzzy logic load balancing service is the core part of the system. Its main function is to analyze information passed from the load monitor, and then make a decision to forward client request to appropriate server. The structure of the fuzzy logic load balancing service is shown in Figure 4. The fuzzy logic load balancing service performs actions such as registering itself to the lookup service and obtaining service proxies available in the lookup service. Thus, it acts as both a server and client as it consists of both the `JoinManager` and `ServiceDiscoveryManager`. Our proposed fuzzy logic load balancing algorithm is implemented in the fuzzy logic controller.

Figure 4: Structure of the fuzzy logic based load balancing service host and the overall load balancing system.

With the above components, the load balancing mechanism can be summarized as follows (illustrated in Figure 5).

1. A client obtains a fuzzy logic load balancing service proxy and sends request to the service.
2. Load monitor asynchronously sends information to fuzzy logic controller for analysis.
3. Fuzzy logic load balancing service periodically measures the remote method invocation time from the load balancer to the server.
4. When a new piece of information such as remote method invocation time or server load arrive at the fuzzy logic controller, the fuzzy inference engine will start to analyze and assign ranks to different servers.
5. After determining which server is the appropriate candidate, the fuzzy logic controller forwards the client request to that server.

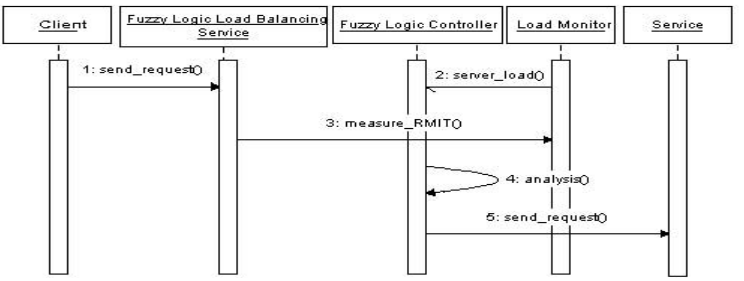


Figure 5: The fuzzy-decision based load balancing mechanism.

3 Performance Results

To evaluate our approach, we have implemented a distributed object platform using Jini and experiments were performed to analyze the client response time and throughput of different load balancing schemes. In order to simulate real client access patterns, a request sequence was generated by using a random number generator to place requests in a given time interval. The request sequences consist of request bursts and intervals of silence. For comparison, we also implemented a system with load balancing algorithm using random and round-robin load distribution by incorporating these algorithms into our fuzzy logic load balancing service. We have set up the testing environment as follows. Six machines are assigned as Jini services and two others are assigned as Jini clients. All the machines are connected by an Ethernet hub with bandwidth of 10Mbps. The configuration of six server machines are: (1) two 500MHz CPU Intel Pentium III workstations, (2) two 667MHz CPU Intel Pentium III workstations, and (3) two 450MHz CPU Intel Pentium III workstations. All machines are equipped with 128MB memory. We have another two machines, which are 600MHz CPU Intel Pentium III workstations with 128MB memory, holding lookup services and fuzzy logic load balancing service. The client machines we used are all 200MHz CPU Pentium with 64MB memory. All machines are running Red Hat Linux 7.0 as their operating systems. Java Development Kit version 1.3 and Jini Technology Starter Kit 1.1 are used to develop all system components. A stateless service, Fibonacci function, is chosen as our benchmark program to simulate consumption of CPU clock cycle in the server machines. Fibonacci function provides a suitable workload for our load balancing tests since each operation can run for a relatively long time. Due to space limitations, only part of the experimental results are included in this paper. More detailed results and analysis can be found in [1].

The average client response times of the three load balancing algorithms as a function of the number of servers are shown in Figure 6, which illustrates that the fuzzy-based approach outperforms the other algorithms consistently for different number of servers. The average client response time of random load balancing algorithm is the highest under all the cases because

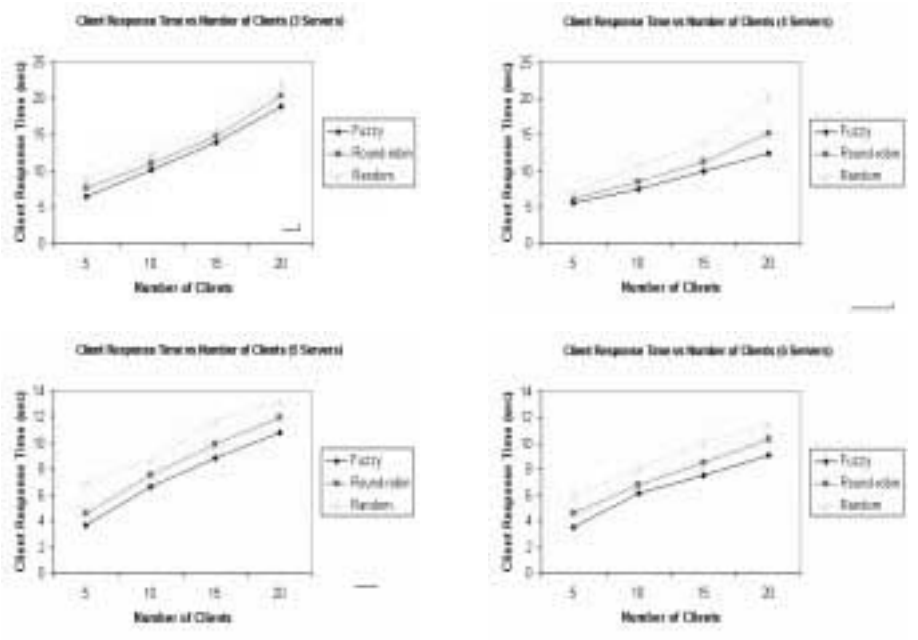


Figure 6: Average response times of the clients.

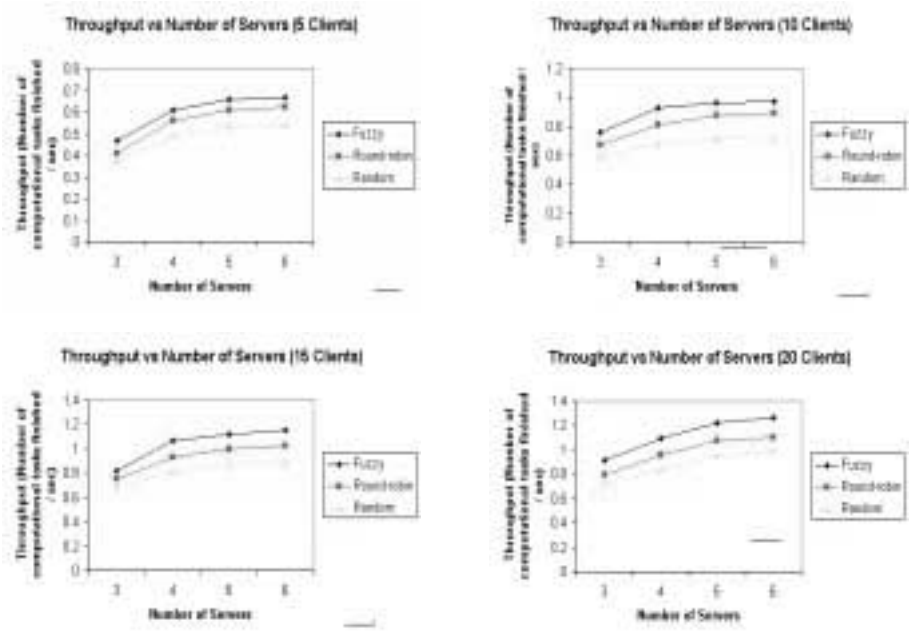


Figure 7: Average throughput of the clients.

uneven distribution of load exists in the random load balancing algorithm. A server with less computing power causes a higher response time when it is suddenly overloaded. This effect deteriorates the overall performance and causes the highest response time. Figure 7 shows how the average throughput differs between each load balancing strategy. In this measurement, 5 to 20 clients were used and each client generated 50 requests. Each client request will generate a computational task using Fibonacci function. The experiment is repeated 100 times for different number of servers. As can be seen from Figure 7, throughput increases as the number of servers increases. Again, the throughput of random load balancing algorithm is the worse among the three algorithms due to the fact than an overloaded computing machines will lengthen the completion time of a task and thus, reducing the overall throughput. For throughput-sensitive application, random load balancing algorithm is not suitable. On the other hand, the throughput of fuzzy-based approach performs the best among the three. The reason is that our approach assigns more requests to the machines with better performance based on fuzzy analysis. This significantly reduces the completion time of a task.

4 Conclusions

Load balancing is an old problem. But new solutions are required in modern distributed object computing platforms, which are increasingly being used in developing many commercial distributed applications. In this paper, we describe the design, implementation, and evaluation of our proposed fuzzy-decision based load balancing algorithm incorporated in a distributed object middleware based on the Jini platform. Our fuzzy-decision load balancer is motivated by the fact that classical and recent load balancing algorithms are inadequate for use in the target platforms considered in our study because there are a multitude of new requirements exist. Our fuzzy-decision load balancing service, based on concise and rather easy to implement rules, is found to be very effective in our extensive experimental studies using a real Jini-based testbed.

References

[1] L.-S. Cheung, *Load Balancing in Distributed Object Computing Systems*, M.Phil. Thesis, Department of Electrical and Electronic Engineering, The University of Hong Kong, May 2001.

[2] C. W. Cheong and V. Ramachandran, "Genetic Based Web Cluster Dynamic Load Balancing in Fuzzy Environment," *Proc. 4th Intl Conf. High Performance Computing in the Asia-Pacific Region*, vol. 2, pp. 714–719, 2000.

[3] E. Damiani, "An Intelligent Load Distribution System for CORBA-Compliant Distributed Environments," *Proc. IEEE Int'l. Conf. Fuzzy Systems*, vol. 1, pp. 331–336, 1999.

[4] M. V. Devarakonda and R. K. Iyer, "Predictability of Process Resource Usage: A Measurement-Based Study on UNIX," *IEEE Trans. Software Engineering*, vol. 15, no. 12, pp. 1579–1586, Dec. 1989.

[5] S. Dierkes, "Load Balancing with a Fuzzy-Decision Algorithm," *Information Sciences*, vol. 97, Issue 1-2, Mar. 1997.

[6] K. K. Goswami, M. Devarakonda, and R. K. Iyer, "Prediction-Based Dynamic Load-Sharing Heuristics," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, no. 6, pp. 638–648, June 1993.

[7] W. Keith, *Core Jini*, Prentice Hall, 1999.

[8] B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice Hall, New Jersey, 1992.

[9] T. Kunz, "The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme," *IEEE Trans. Software Engineering*, vol. 17, no. 7, pp. 725–730, July 1991.

[10] P. Mehra and B. Wah, "Synthetic Workload Generation for Load-Balancing Experiments," *IEEE Parallel and Distributed Technology*, pp. 4–19, 1995.

[11] R. Mirchandaney, D. Towsley, and J. A. Stankovic, "Analysis of the Effects of Delays on Load Sharing," *IEEE Trans. Computers*, vol. 38, no. 11, pp. 1513–1525, Nov. 1989.

[12] M. Mitzenmacher, "How Useful Is Old Information?," *IEEE Trans. Parallel and Distributed Systems*, vol. 11, no. 1, pp. 6–20, Jan. 2000.

[13] A. Shaout and P. McAuliffe, "Job Scheduling Using Fuzzy Load Balancing in Distributed System," *Electronics Letters*, vol. 34, no. 20, pp. 1983–1985, Oct. 1998.

[14] N. G. Shivaratri, P. Krueger, and M. Singhal, "Load Distributing for Locally Distributed Systems," *Computer*, vol. 25, no. 12, pp. 33–44, Dec. 1992.