A Quantitative Comparison of Load Balancing Approaches in Distributed Object Computing Systems

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†

Several load balancing schemes are recently proposed for distributed object computing systems, which 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. However, while the rationales and mechanisms employed are dramatically different, the relative strengths and weaknesses of these approaches are unknown, making it difficult for a practitioner to choose an appropriate approach for the problem at hand. In this paper, we describe in detail three representative approaches, which are all practicable, and present a quantitative comparison using our experimental distributed object computing platform. Among these three approaches, namely, JavaSpaces based, request redirection based, and fuzzy decision based, we find that the fuzzy decision based algorithm outperforms the other two considerably.

Keywords: distributed object computing, load balancing, Java, JavaSpaces, Jini, fuzzy decision.

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 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, a load balancing technique can be introduced to distribute workload in a judicious manner among various machines [14], [11], [20], [21].

Numerous approaches are suggested for performing load balancing in distributed computing systems [13], [15], [17], [18], [19]. In this paper, we describe in detail three representative approaches: (1) a JavaSpaces approach, (2) a request redirection approach, and (3) a fuzzy decision based approach. The JavaSpaces approach [1], [2], [3], [10], [12], works by using an Java object space as a task pool, from which the servers get work for execution. Load balancing is thus achieved in an uncoordinated manner. In the request redirection approach, a threshold is set in each server such that when the number of outstanding requests exceed the threshold, the new requests are redirected to other servers in a round-robin manner. Due to space limitations, these two approaches are not elaborated here but details can be found in [15]. Proposed recently in [16], the fuzzy decision based approach works by using fuzzy decision variables to capture the inherent dynamic behaviors of the system states in order to make intelligent load balancing decisions. Our proposed system is based on Jini [4], [5], [6]. More details about this novel approach is described in Section 2. Implemented on the same experimental object computing network comprising a number of Pentium workstations, all the approaches are evaluated extensively under a wide range of parameters. The experimental results, described in Section 3, indicate that the fuzzy based approach outperforms the other two approaches considerably. We conclude the paper in Section 4.

2 Fuzzy Logic Based Request Redirection Approach

In this section, we describe our recently proposed fuzzy logic based request redirection scheme [15], [16]. In this approach, a fuzzy logic controller is incorporated into the request redirection system discussed in the previous section. The role of the fuzzy logic controller is to control the request redirection decision of each server. Instead of manually set the

† 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.

0-7695-1372-7/01 $10.00 © 2001 IEEE

threshold value for each server, the fuzzy logic mechanism allows servers to make request transfer decision based on the server ranking assigned by the fuzzy logic controller. In the following, a brief introduction of using fuzzy logic controller is presented, followed by the description of our design model and implementation.

2.1 Fuzzy Logic Controller

To tackle the load balancing problem, conventional control theory can be applied to restore system equilibrium. For instance, a sudden increase in client requests can be modeled as external force which attempts to destabilize the network and the end-hosts. The stability of the network can be maintained by using feedback control which performs suitable adjusting actions to minimize the effect of the external force. In order to design a load balancing algorithm based on conventional control theory, one has to develop a mathematical model of the system to be controlled and determine the characteristics of the model by applying various analytical techniques. Indeed, with the incorporation of some simplifying assumptions, model with linear relationships between a few variables can be easily set up. However, 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. This limitation makes it very difficult, if not intractable, to apply conventional control theory to balance load in computer network. To overcome this problem, fuzzy logic control theory [7] 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 [7]. 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 [7].

The architecture of the fuzzy logic controller 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.

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, `high` or `low` can be used in the variable `server_load`. These linguistic values are modeled as fuzzy sets. Based on the linguistic values, their corresponding membership functions can be expressed based on application requirements.

2.2 Design and Implementation of Fuzzy Logic Based Request Redirection Approach

In the our proposed approach, fuzzy logic is used to help in decision making for request redirection. Fuzzy logic is implemented inside a fuzzy logic controller of which the main function is in processing all server load information and assigning different ranks to servers. Each server can based on its own server rank and other servers' ranks to decide where it should transfer a remote call to. As the name implies, fuzzy logic controller has to make use of fuzzy information so as to perform logic control. Several linguistic variables, server load, server load mean deviation, and server rank, are used in the fuzzy logic algorithm and are defined as follows.

2.2.1 Server Load

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 [8], [9]. 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.

2.2.2 Server Load Mean Deviation

In order to determine whether a server should redirect its request to other servers or not. The deviation of each server load (SLMD) from the mean server load is calculated. The deviation is defined as follows where i is the server number.

$$SLMD_i = SL_i - \overline{SL}$$

The mean value of server load is updated every time a server load information arrives to the fuzzy logic controller. Server Load Mean Deviation (SLMD) is defined as: `{negative(N), zero(Z), positive (P)}`.

2.2.3 Service Rank

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 higher the rank that a service gets, the more appropriate that it should redirect request to other servers. After defining the above fuzzy variables, a set of inference rules is defined.

By applying the fuzzy inference rules, a decision can be generated based on both antecedents. That is, if SLMD is positive and SL is high, 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 SLMD 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 [7], 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 [7] to defuzzify the output. The overall centroid of N overlapping areas A_i for $i = 1, 2, \ldots, N$ is given by:

$$\bar{x} = \frac{\sum_{i=1}^{N} \bar{x}_i A_i}{\sum_{i=1}^{N} A_i}$$

where A_i and $\bar{x}_i$ are the overlapped area and centroid from the triangles or trapeziums obtained in the i-th rule, respectively. The centroid and area are calculated for each triangles or trapeziums. This process is repeated for other inference rules where the inputs are applied to obtain an area composed of overlapped trapeziums. The defuzzification process generates a centroid value which represents the rank of a service. The higher the service rank, the more appropriate that the server should redirect request. In other words, a server with lower service rank is an appropriate candidate to receive extra requests. To summarize, Figure 1 shows the setup of the fuzzy logic request redirection approach.

The configuration of our approach is similar to the previous approach except that there exists a fuzzy logic controller which collects server load information from the load monitor of each server machine. After the fuzzy inference process, service rank information is multicasted by the fuzzy logic controller such that servers can know each other service ranks. A server redirects its incoming request if its service rank is the highest. The server seeks a target server service with the lowest service rank. The load monitors multicast their server load value in every two seconds. If a UDP multicast packet is received by the fuzzy logic controller, the fuzzy logic controller decodes the packet, analyses the respective service rank, and multicasts packets in every five seconds.

3 Performance Results

To quantitatively evaluate the three different approaches, we have implemented a distributed object platform based on Java 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.

3.1 Hardware/Software Platform

We have set up the testing environment consisting of several Pentium PCs. 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

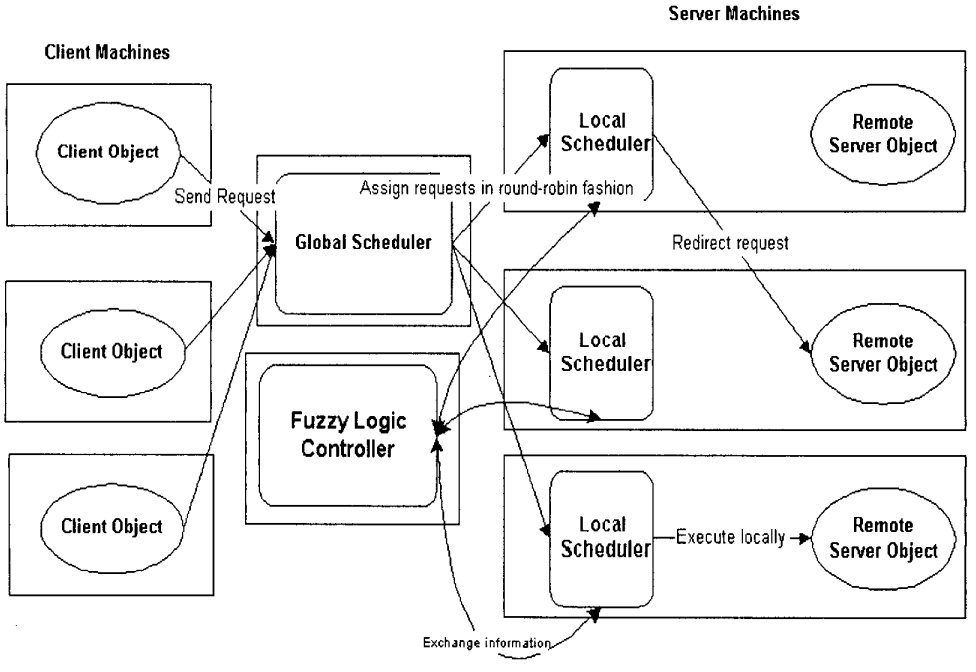


Figure 1: Fuzzy logic request redirection.

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.

3.2 Load Distributed Results of JavaSpaces Appraoch

In the JavaSpaces-based load distribution approach, in order to exploit the power of JavaSpaces as much as possible, we run a JavaSpaces service in a symmetric multiprocessors (SMP) machine with four 450MHz CPU Intel Xeon Pentium III and 1 GB memory. The large memory capacity of the SMP minimizes the chance of occurring JavaSpaces out of memory exception. The configuration of other server and client machines are kept the same. In our context, we define execution length as the time difference between the time just before the first client writes a request and the time immediately after the last client takes a response.

As can be seen from the load distribution of the JavaSpaces approach shown in Figure 2(a) and Figure 3(a), the benchmark readings of all servers remain steady during the execution when the number of clients is 20. The rate of adding the tasks into the JavaSpaces service by the clients is faster than the rate of retrieving the tasks from the JavaSpaces service by the servers. A server can immediately pick up the next task after finishing the previous one. There is no need for the server to wait before taking up another task. Thus, the benchmark readings of all servers remain steady as the servers are busy all the time.

As compared with other approaches such as Round-robin and Random, the total execution time of the JavaSpaces approach is relatively longer. The total execution time depends on factors such as the number of clients, the time interval between two requests, the program overheads, and the design of system. The long execution time of the JavaSpaces approach is due to two factors: (1) the cost of remote method invocation, and (2) the single thread implementation of servers. It is obvious that there involves four remote method invocations for a task to get done. A client has to wait at least four remote invocation time before it can issue another request. Moreover, since the servers are designed as single thread, they will only pick up one task at a time. There is no parallel execution of tasks within a machine. Servers have to get tasks from the space one by one as compared with other approaches in which remote server objects are multithreaded and accept several remote method invocation requests from clients at the same time. Thus, the overall execution time of JavaSpaces approaches is lengthened. One possible way to shorten the execution time is to start several worker processes in a more powerful machine so that it can take more than one task at the same time.

3.3 Load Distribution Results of Request Redirection Approach

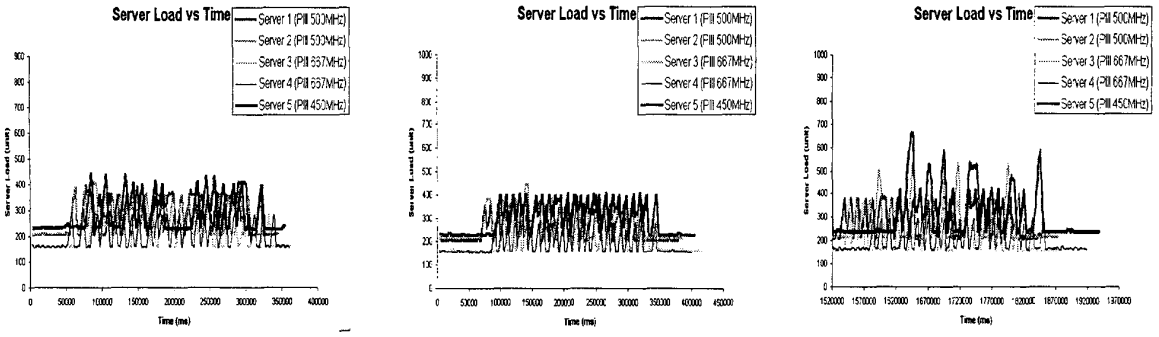
In the request redirection approach, requests are distributed in a round robin fashion and are redirected to other server if the load index (the number of concurrent requests) exceeds a threshold value predefined and there exists an available receiver. Figure 2(b) and Figure 3(b) show the server load distribution using request redirection approach with 5 and 20 clients, respectively. It can be observed that the shape of the benchmark readings of this approach is similar to the round-robin approach. The notable difference is that the lowest processing power machines (server 5 and server 6) now sustain a lower load as compared with the pure round-robin algorithm during the execution lifetime. It is because the request redirection mechanism transfers part of the workload from server 5 and server 6 to server 1 and server 2.

It can be seen that there is a significant improvement using round-robin algorithm with request redirection mechanism over traditional round-robin in heterogeneous computing environment. Since requests can be redirected to the more powerful servers if required. However, it should be noted that the success of this algorithm is not only due to the request redirection mechanism but also the determination of threshold value for each server under different client conditions. As mentioned before, the threshold value of each server is manually set based on the performance data obtained from the round-robin approach. It would be interested to explore other means to control the redirection mechanism such that the overall system can adapt itself to the environment flexibly.

3.4 Load Distributed Results of Fuzzy Logic Based Request Redirection Approach

In fuzzy logic based request redirection approach, requests are distributed in a round robin fashion and are redirected based on service rank. Figure 2(c) and Figure 3(c) show the server load distribution using fuzzy logic based request redirection approach with 5 and 20 clients, respectively.

It can be seen that the server benchmark readings fluctuate vigorously as compared with round-robin algorithm. It is because when a server attains the highest service rank among the servers, it will initiate a load transfer when a new incoming request arrives. The server load will then drop and it will redirect request again when its service rank become the highest. Thus, a heavily loaded machine can intelligently redirect its load to the underloaded machine based on the service rank. A more powerful machine can now accept more requests which is sent from a less powerful machine. It is due to this redirection mechanism which makes the amplitude of server load fluctuation larger as compared with the previous approach in which a server will maintain its server load around a predefined threshold value. Since the performances of the two approaches are similar, it should be noted that the fuzzy logic controller approach is more flexible than the threshold based redirection approach. It is because the threshold value is set manually based on performance data predetermined in round-robin algorithm. If there is a sudden increase in client requests, the threshold value has to be adjusted accordingly. In this approach, the fuzzy logic controller will send updated service

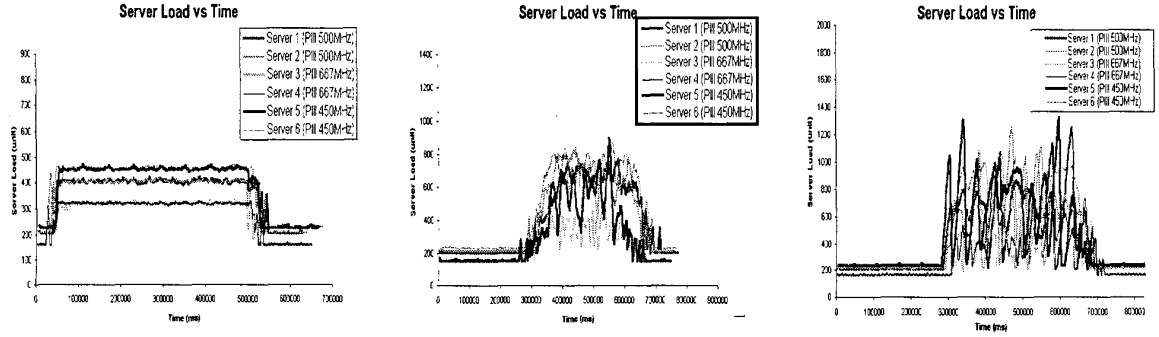


(a) JavaSpaces Based Approach

(b) Request Redirection

(c) Fuzzy Decision Based Request Redirection

Figure 2: Load distribution of the three approaches for 5 servers and 5 clients.



(a) JavaSpaces Based Approach

(b) Request Redirection

(c) Fuzzy Decision Based Request Redirection

Figure 3: Load distribution of the three approaches for 6 servers and 20 clients.

rank information to servers no matter how the system load varies. Therefore, the fuzzy logic controller approach can adapt itself to the changing of environment.

3.5 Average Client Response Time and Throughput

The average client response times of six load balancing algorithms (pure fuzzy [16], round-robin, random, request redirection, fuzzy based request redirection, and JavaSpaces) as a function of the number of servers are shown in Figure 4(a), which illustrates that the fuzzy-based approach outperforms the other algorithms consistently for different number of servers. The average client response time of JavaSpaces-based and random load balancing algorithm is comparatively higher than other algorithms under all the cases because uneven distribution of load exists in the random load balancing algorithm and the single threaded worker structure in the JavaSpaces approach. In random load balancing, 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 4(b) shows how the average throughput differs between each load balancing strategy. In this measurement, 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 4(b), throughput increases as the number of servers increases. Again, the throughput readings of random load balancing algorithm and JavaSpaces are worse than the other four algorithms. In random load balancing, it is due to the fact than an overloaded computing machines will lengthen the completion time of a task and thus, reducing the overall throughput. In JavaSpaces approach, the poor throughput performance is also due to the single threaded design of service object. If more workers are started in a machine, the overall throughput will be improved. 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 six. The reason is that our approach assigns more requests to the machines with better performance based on fuzzy analysis.

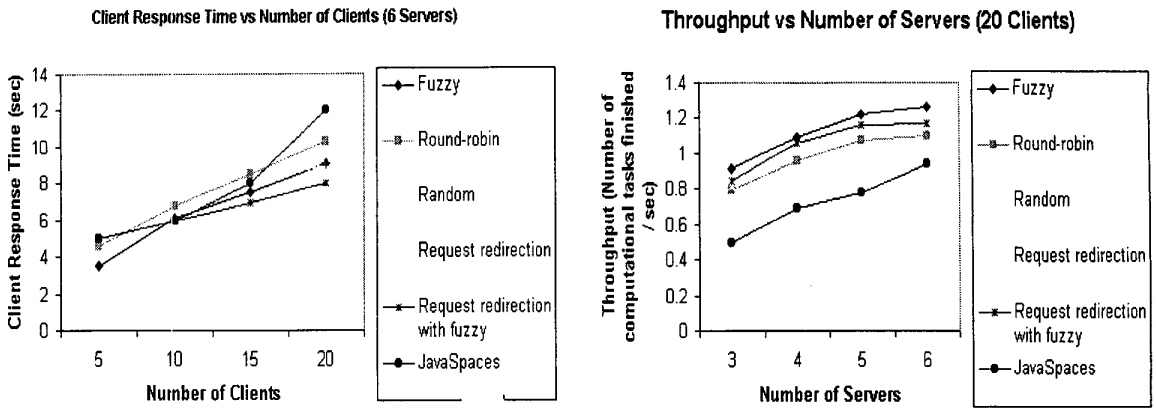


Figure 4: Average response times and throughput of the clients.

This significantly reduces the completion time of a task. Since the two request redirection approaches achieve similar throughput performance results, one can choose a request redirection algorithm based on its complexity and adaptability.

4 Conclusions

In this paper, we have described three contemporary load balancing approaches for distributed object computing systems. The first approach uses JavaSpaces service as a task pool for servers to get tasks for execution. The second approach uses a round-robin request redirection mechanism. The third approach uses a fuzzy decision based approach to perform the request redirection. We have implemented the three approaches on a common experimental object computing system based on Java using a number of Pentium PCs. The experimental results indicated that the fuzzy based approaches are more robust and flexible, and outperform other approaches considerably.

References

[1] E. Freeman, S. Hupfer, and K. Arnold, *JavaSpaces: Principles, Patterns, and Practice*, Addison-Wesley, 1999.

[2] Hosta White Paper, Concept Technologies Ltd., 2000.

[3] IBM's TSpaces, `http://www.almaden.ibm.com/cs/TSpaces`.

[4] Jini Connection Technology, Jini Technology Core Platform Specification, `http://www.sun.com/jini/specs/jini1_1spec.html`.

[5] Jini Connection Technology, JavaSpaces Service Specification, `http://www.sun.com/jini/specs/jini1_1spec.html`.

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

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

[8] 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.

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

[10] OpenSpaces, `http://openspaces.exolab.org/`

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

[12] S. Ahuja, N. J. Carriero, and D. H. Gelernter, "Matching Language and Hardware for Parallel Computation in the Linda Machine," *IEEE Trans. Computers*, vol. 37, no. 8, pp. 921–929, Aug. 1988.

[13] V. Cardellini, M. Colajanni, and P. S. Yu, "Dynamic Load Balancing on Web-Server Systems," *Internet Computing*, May/June 1999, pp. 28–39.

[14] T. L. Casavant and J. G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems," *IEEE Trans. Software Engineering*, vol. 14, no. 2, pp. 141–154, Feb. 1988.

[15] L.-S. Cheung, *Load Balancing in Distributed Object Computing Systems*, M.Phil. Thesis, Department of EEE, The University of Hong Kong, May 2001.

[16] L.-S. Cheung and Y.-K. Kwok, "A Fuzzy Load Balancing Service for Network Computing Based on Jini," *Proc. EURO-PAR'2001*, Manchester, United Kingdom, August 2001.

[17] 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.

[18] P. Chulhye and J. G. Kuhl, "A Fuzzy-Based Distributed Load Balancing Algorithm for Large Distributed Systems," *Proc. 2nd Int'l. Sym. Autonomous Decentralized Systems*, pp. 266–273, Apr. 1995.

[19] C.-J. Hou and K. G. Shin, "Implementation of Decentralized Load Sharing in Networked Workstations Using the Condor Package," *Journal of Parallel and Distributed Computing*, vol. 40, pp. 173–184, 1997.

[20] A. Leff and P. S. Yu, "A Performance Study of Robust Distributed Load Sharing Strategies," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 12, pp. 1286–1301, Dec. 1994.

[21] J. Watts and S. Taylor, "A Practical Approach to Dynamic Load Balancing," *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 3, pp. 235–248, Mar. 1998.