Fuzzy Decision Load-balancing Algorithm for Cloud-based Terminal Services

Hsin-Hung Chen[1], Li-Shing Huang[1], Jian-Bo Chen[2], Tsang-Long Pao[1]

[1] Tatung University, Department of Computer Science and Engineering, Taiwan
[2] Ming Chuan University, Department of Information and Telecommunications Engineering, Taiwan
{hsinhung.chen, joeintw}@gmail.com, jbchen@mail.mcu.edu.tw, tlpao@ttu.edu.tw

Abstract

In this paper, we propose a desktop-as-a-service system that can be used on a campus to enable full utilization of powerful hardware and software resources. Our proposed system comprises Microsoft's terminal services and a dispatcher. The dispatcher provides a single portal and coordinates with different terminal services as part of the overall desktop-as-a-service system. When a client issues a request, the dispatcher uses a fuzzy decision algorithm to balance the load between terminal services. The fuzzy decision algorithm comprises three steps, namely, fuzzification, rule evaluation, and defuzzification. Based on these steps, the final crisp values can be calculated. As such, the terminal service with the highest crisp value is the most appropriate for serving the given client request. Our experimental results show that our fuzzy decision algorithm successfully achieves the highest possible performance.

Keywords: Fuzzy decision, Load balance, Cloud Terminal services

1 Introduction

The concept of a cloud system emphasizes that anyone, anywhere, can use any tools to connect to the Internet, obtaining the same services and achieving the same results. Based on this concept, we build a cloud system that can provide users with legally licensed software without installing it on their own computer. To accomplish this, we built a cloud system based on Microsoft's terminal service [1-2]. In our system, we install legally licensed software in the terminal server. Next, users can use the client software of our terminal service to connect to the cloud-based terminal service system and use legally licensed software [3].

Given that resources of one terminal server (i.e., CPU, memory, network, etc.) are shared, if multiple users connect to the same terminal server, they compete for these resources [4]. Moreover, a single terminal server has a limited number of connections. If the given resources are exhausted, a new user cannot be served by the given terminal server. To overcome this problem, we must build terminal servers to provide services to more and more users; however, when the cloud system comprises several terminal servers, effectively dispatching user requests to the most appropriate terminal server immediately becomes a crucial issue [5].

The first solution for handling multiple terminal servers is to create multiple individual portals that the user can choose from regardless of the current load of each terminal server. This method is simple, but it does not provide any measure of load balancing. The second solution here is to use a dispatcher as the portal [6]. Requests from users are connected to this single dispatcher. The dispatcher then uses a load-balancing algorithm to select the most appropriate terminal server, providing this information to the user such that the user can connect to the terminal server with least load.

Several load-balancing algorithms can be adopted in our desktop-as-a-service system [7-8]. The simplest algorithms are random and round-robin algorithms. The random algorithm chooses the most appropriate target simply by random, which does not consider any load information regarding the terminal servers. The round-robin algorithm chooses each terminal server in turn, which also does not consider load information regarding the terminal servers.

The least connection algorithm chooses the most appropriate terminal server according to the current number of established connections. If one terminal server has fewer users than all others, this terminal server will be the most appropriate one to serve the user. This algorithm can balance load in some situations, but the least connection algorithm still does not consider other factors of the terminal servers. If one user connects to a terminal server and runs a CPU-bound multimedia application, the CPU load and memory usage will be substantially higher than others, but the connection will still count as one connection.

To avoid this problem, we propose incorporating additional features in making the dispatcher decision

*Corresponding Author: Hsin-Hung Chen; E-mail: hsinhung.chen@gmail.com

DOI: 10.3966/160792642018051903007

[9]. The features we choose here are CPU idle percentage, available memory, available bandwidth, and available number of connections [10]. Based on these four features, we use fuzzy decision to calculate the final crisp value [11-13]. Our fuzzy decision algorithm consists of three steps, namely fuzzification, rule evaluation, and defuzzification. We use the four features noted above as input parameters, then we normalize these four features. We define four membership functions for each parameter by which membership function values can be obtained. We then pass these membership function values into the rule base for rule evaluation. Results of rule evaluation consist of four decision values. Finally, we use these four decision values to calculate the final crisp values. The terminal server with the highest crisp value is then deemed the most appropriate one to serve that user at that given time.

In addition to this introduction, our paper is organized as follows. In Section 2, we introduce our desktop-as-a-service system. In Section 3, we describe traditional load-balancing algorithms. In Section 4, we introduce the features selected in this paper, then in Section 5, we describe our proposed fuzzy decision algorithm. In Section 6, we describe our experimental environments and results. Finally, in Section 7, we provide conclusions and directions for future work.

2 Desktop-as-a-service System

There are many kinds of cloud services available. One such cloud service provides users with software, i.e., users do not need to install software and instead connect to a cloud service system and execute any pre-installed software on the cloud. Such software includes the Microsoft Office series, the Virtual Studio series, Adobe Creative Suite, and so on. Some of these software packages are limited in terms of the number of licensed users, thus we cannot provide everyone a copy of the software. Instead, we install the software in a cloud system, and users who wish to use the software can connect to this cloud system.

To implement this kind of cloud system, we adopt Microsoft's terminal service, which provides users a means of establishing a connection to the terminal service via the Remote Desktop Protocol (RDP) [2]. When a user connects to the terminal service, all programs the user executes are on the terminal service site. The user site merely shows the desktop screen. The CPU and memory are provided by the terminal service. In RDP, even the user machine is poor specification, the user also can execute high-resource applications, such as multimedia applications. The architecture of this terminal service is shown in Figure 1.

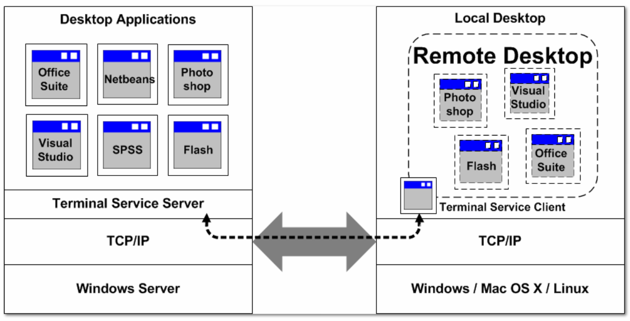


Figure 1. Terminal service architecture

3 Traditional Load-Balancing Algorithms

It is difficult to use one terminal service to support many users. We therefore propose building multiple terminal services that coordinate with one another, thus forming a cluster. In this cluster, we also require a load-balancing algorithm that balances load between each of the terminal services. Figure 2 shows the general architecture of our load-balancing system.

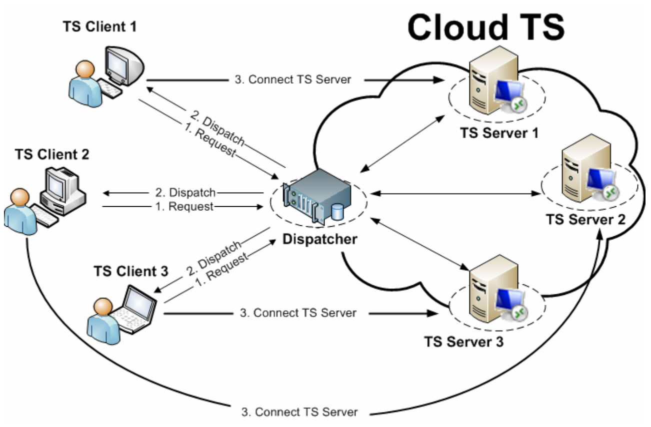


Figure 2. Dispatcher-based load-balancing architecture

Traditional load-balancing algorithms include the random, round-robin, and least connection algorithms. In the random algorithm, the dispatcher randomly chooses a terminal service and transmits its domain name or IP address to the requesting user. The user then establishes a connection to the given terminal service. In the round-robin algorithm, the dispatcher chooses each terminal service in turn, i.e., the first user is served by the first terminal service, the second user is served by the second terminal service, and so on. Both the random and round-robin algorithms do not consider any load information of the terminal services, thus the entire system cannot evenly share the load. Nonetheless, these two algorithms are easy to implement.

The third traditional load-balancing algorithm we introduce here is the least connection algorithm. In this algorithm, the dispatcher must keep track of the number of connections at each of the terminal services. When a new user issues a request, the dispatcher chooses the terminal service with the lowest number of connections. Although this algorithm considers the number of connections, this algorithm does not consider other important factors that influence the overall performance of the system. For example, if one user connects to a terminal service and executes a three-dimensional multimedia application, the allocated resources, including CPU and memory, will be exhausted by this single user, even though the number of connections to this terminal service may be the lowest.

In our proposed fuzzy decision load-balancing algorithm, we first identify the features. Next, based on these features, we execute the three steps of the fuzzy decision algorithm, namely fuzzification, rule evaluation, and defuzzification. Finally, the resulting crisp values are used to decide which terminal service is most appropriate for the given user request.

4 Feature Selection

Some studies have described how CPU load and the number of user connections impact overall system performance [14]. Further, some studies used CPU load, memory size, and I/O throughput characteristics to measure server capacity [15-16]; cloud computing also requires the inclusion of network utilization and the number of current connections [17-18]. Therefore, we selected four factors of CPU idle percentage, memory available, bandwidth usage and the number of current connections as features in our fuzzy decision algorithm. Using Microsoft's terminal service, we implemented the four functions shown in Figure 3 to Figure 6 to calculate the four feature values.

```
public static string CPU_Loading() {
   ManagementObjectSearcher MOS =
      new ManagementObjectSearcher("root\\CIMV2",
         "SELECT * FROM Win32_PerfFormattedData_PerfOS_Processor
            where Name = '_Total'");
      foreach (ManagementObject MO in MOS.Get()) {
         ReturnValue = MO["PercentIdleTime"].ToString();
      }
      return ReturnValue;
}
```

Figure 3. Function to retrieve CPU idle percentage

```
public static string Memory_Usage() {
   ManagementObjectSearcher MOS =
      new ManagementObjectSearcher("root\\CIMV2",
         "SELECT * FROM Win32_PerfFormattedData_PerfOS_Memory");
   foreach (ManagementObject MO in MOS.Get()){
      ReturnValue = MO["AvailableBytes"].ToString();
   }
   return ReturnValue;
}
```

Figure 4. Function to retrieve available memory

```
public static string Network_Utilization(string NIC_NAME){
   ManagementObjectSearcher MOS =
      new ManagementObjectSearcher("root\\CIMV2",
         "SELECT * FROM Win32_PerfFormattedData_Tcpip_NetworkInterface
            where Name = '" + NIC_NAME + "'");
   foreach (ManagementObject MO in MOS.Get()){
      ReturnValue = MO["BytesTotalPersec"].ToString();
   }
   return ReturnValue;
}
```

Figure 5. Function to retrieve bandwidth usage

```
public static string Session_Count() {
   ManagementObjectSearcher MOS =
      new ManagementObjectSearcher("root\\CIMV2",
         "SELECT * FROM Win32_TerminalService");
   foreach (ManagementObject MO in MOS.Get()){
      ReturnValue = MO["TotalSessions"].ToString();
   }
   return ReturnValue;
}
```

Figure 6. Function to retrieve the number of connections

5 Fuzzy Decision Load-Balancing Algorithm

In this section, we describe our fuzzy decision load-balancing algorithm. After we select the features, the decision-making is determined via fuzzy logic. Here, fuzzy logic consists of three steps, i.e., fuzzification, rule evaluation, and defuzzification. The processes that comprise our fuzzy decision algorithm are shown in Figure 7.

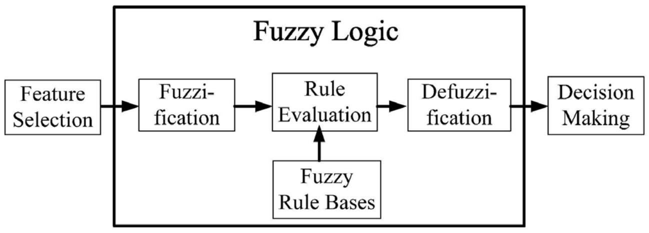


Figure 7. The processes comprising our fuzzy decision load-balancing algorithm

As noted above, we selected four features as specific input parameters. After the fuzzification step, we obtain eight membership function values based on corresponding membership functions. The number of combinations of these eight values is 16, and the four fuzzy decision values are obtained based on the fuzzy rule base. After the defuzzification step, the final crisp values are obtained. The fuzzy logic mechanism is shown in Figure 8.

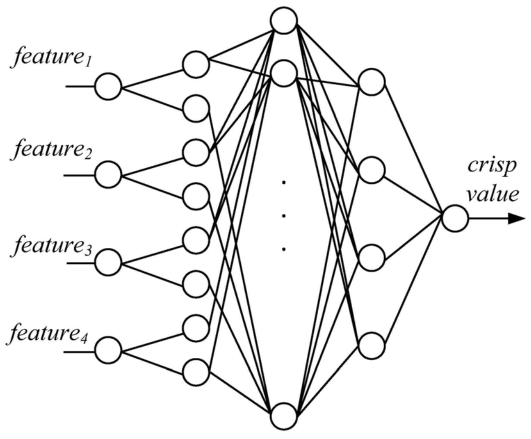


Figure 8. Steps of the fuzzy logic process

5.1 Fuzzification

Fuzzification is the process of converting input parameters into relevant fuzzy membership values according to corresponding membership functions. For each feature, we define two membership functions. These two membership functions are the high and low membership functions. For each feature, we therefore obtain two membership values.

The four features selected are CPU idle percentage (*CPU*), available memory (*MEM*), available bandwidth (*BWD*) and available connection number (*CON*). Two membership functions are defined for each feature. They are *HCPU* and *LCPU* for the feature *CPU*, *HMEM* and *LMEM* for the feature *MEM*, *HBWD* and *LBWD* for the *BWD* and *HCON* and *LCON* for the feature *CON*. These membership functions are defined as per equations (1) and (2) and are shown in figures Figure 9 to Figure 12.

$$f(x) = x, \quad for\ 0 \le x \le 1 \tag{1}$$

$$f(x) = 1 - x, \quad for\ 0 \le x \le 1 \tag{2}$$

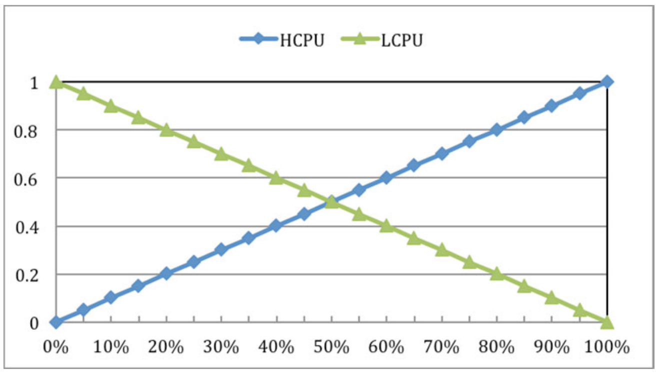


Figure 9. Membership functions of *HCPU* and *LCPU*

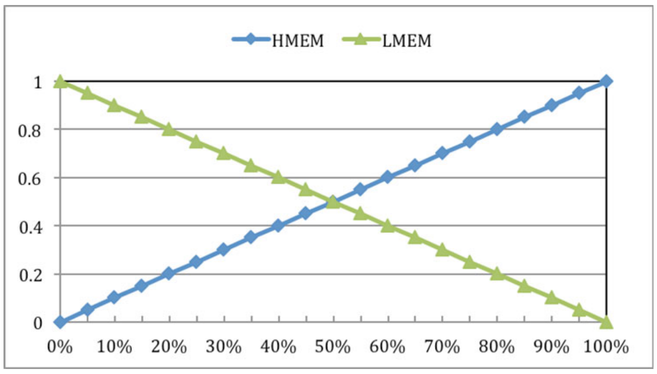


Figure 10. Membership functions of *HMEM* and *LMEM*

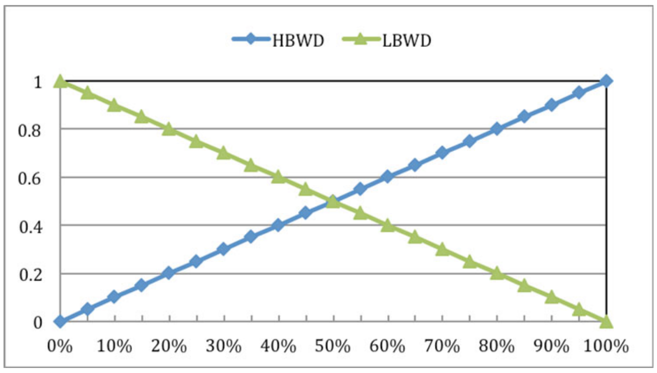


Figure 11. Membership functions of *HBWD* and *LBWD*

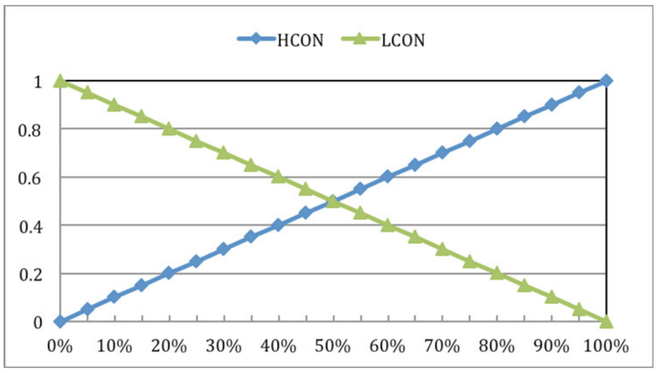


Figure 12. Membership functions of *HCON* and *LCON*

5.2 Rule Evaluation

The main purpose behind rule evaluation is to apply membership function values to the rule base to obtain resulting rule evaluation values. Table 1 shows the rule base in which influence rules are illustrated. In the table, *Y*, *PY*, *PN*, and *N* indicate *Yes*, *Probably Yes*, *Probably No*, and *No*, respectively, all four of these corresponding to membership function values of the four features. Rule evaluation values can be assigned minimum, maximum, or average values for the membership functions of the rules.

Table 1. Fuzzy rule base

CPU	MEM	BWD	CON	Rule Evaluation
HCPU	HMEM	HBWD	HCON	Y
HCPU	HMEM	HBWD	LCON	Y
HCPU	HMEM	LBWD	HCON	Y
HCPU	HMEM	LBWD	LCON	PY
HCPU	LMEM	HBWD	HCON	Y
HCPU	LMEM	HBWD	LCON	PY
HCPU	LMEM	LBWD	HCON	PY
HCPU	LMEM	LBWD	LCON	PN
LCPU	HMEM	HBWD	HCON	PY
LCPU	HMEM	HBWD	LCON	PN
LCPU	HMEM	LBWD	HCON	PN
LCPU	HMEM	LBWD	LCON	PN
LCPU	LMEM	HBWD	HCON	PN
LCPU	LMEM	HBWD	LCON	N
LCPU	LMEM	LBWD	HCON	N
LCPU	LMEM	LBWD	LCON	N

Since each factor has two membership functions, we have 16 possible combinations. Each rule evaluation value in Table 1 has a decision from among *Y, PY, PN*, and *N*. The fuzzy values are used to evaluate rules for obtaining *Fuzzy Decision Values* (*FDVs*) by assigning the average value of the degree of membership of the rules. In this way, the decision and the corresponding *FDV* can be precisely determined.

5.3 Defuzzification

In the defuzzification step, we assign a set of weighted values to the four decision values (i.e., *Y*, *PY*, *PN*, and *N*). Each value represents a different set of weights. Therefore, the Crisp Value (*CV*) can be determined based on the weighted values and the degrees of the fuzzy decision values. The *CV* is calculated as

$$CV = \frac{\sum_i FD(i) \times w(i)}{\sum_i w(i)} \tag{3}$$

where *FD(i)* is the fuzzy decision value of *FD(Y), FD(PY), FD(PN), and FD(N) and w(i)* is the weighted values of the four fuzzy decision values. Each terminal server calculates its own *CV*. The load-balancing algorithm then uses the *CVs* to determine which terminal server is the most appropriate one to serve client requests. More specifically, the terminal server with the highest *CV* is the selected one such that the entire desktop-as-a-service system can achieve the highest level of efficiency.

6 Experimental Environment and Results

In this paper, we proposed a fuzzy decision-based load-balancing algorithm for our desktop-as-a-service system. We present our experimental environment and results in this section. Performance comparisons are also described.

6.1 Experimental Environment

The load-balanced desktop-as-a-service system consists of three key components, namely clients, terminal servers, and a dispatcher. In our experimental environment, we implemented eight terminal servers and one dispatcher. These eight terminal servers have the same hardware specifications, which are summarized in Table 2; specifications for the dispatcher are summarized in Table 3.

Table 2. Specifications of the terminal servers

	Number	Specification
CPU	8	Intel Xeon E5530 2.4GHz
Memory	48GB	DDR3 1333MHz
Network Adapter	1	Broadcom BCM5709S NetXtreme II Gigabit Ethernet
Operating System	1	Windows Server 2008 R2
Terminal Service	200	Terminal Service Licenses

Table 3. Specification of the dispatcher

	Number	Specification
CPU	2	Intel Xeon CPU 3.06GHz
Memory	2GB	DDR2
Network Adapter	1	Broadcom 5703 10/100/1000
Operating System	1	FreeBSD 8.0
		Apache/2.2.14
Software	1	PHP/5.2.12
		MySQL/5.0.90

The processes involved in our load-balanced desktop-as-a-service system are described below and illustrated in Figure 13. First, the client uses a Web browser to connect to our desktop-as-a-service system, i.e., to the dispatcher. After the client account and password are authenticated, the dispatcher issues requests to the terminal services to obtain current load information. When the dispatcher collects all four features from each of the terminal services, the fuzzy decision algorithm is used to calculate crisp values for each terminal services. The terminal service with the highest crisp value is then selected. Next, the dispatcher dynamically generates and sends an RDP file to the client; the RDP file contains the IP address or domain name of the selected terminal service. The client then uses this RDP file to connect.

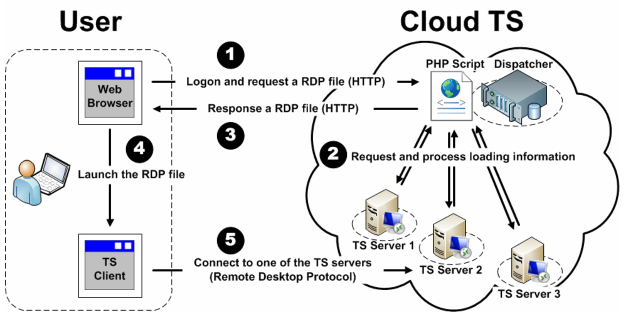

User
Cloud TS
Web Browser
1
Logon and request a RDP file (HTTP)
PHP Script
Dispatcher
Response a RDP file (HTTP)
3
4
Launch the RDP file
2
Request and process loading information
TS Client
5
Connect to one of the TS servers (Remote Desktop Protocol)
TS Server 1
TS Server 2
TS Server 3

Figure 13. The process of connecting to our desktop-as-a-service system

6.2 Experimental Results

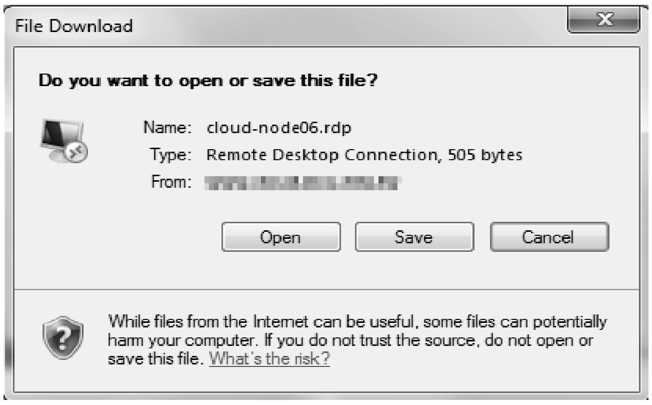
For the dispatcher, we used the Apache webserver and PHP to implement user authentication, collecting load information, fuzzy decision-making, and RDP file generation. The actual implementation for how to generate the RDP file using PHP is shown in Figure 14; how the client can access the RDP file is shown in Figure 15.

```
$rdpserver = Get_RDP_Srv();
$username = $_SESSION["username"];
$filename = "cloud-$rdpserver.rdp";

$str = <<<EOD
domain:s:cloud
username:s:$username
full address:s:$rdpserver
EOD;

header('Content-type: application/x-rdp');
header('Content-Disposition: attachment; filename="'.$filename.'"');
header('Content-Length: '.strlen($str));
print($str);
```

Figure 14. The PHP code to generate the RDP file


File Download

Do you want to open or save this file?

Name: cloud-node06.rdp
Type: Remote Desktop Connection, 505 bytes
From: [redacted]

While files from the Internet can be useful, some files can potentially harm your computer. If you do not trust the source, do not open or save this file. What's the risk?

Figure 15. How the client is able to access the dynamically generated RDP file

6.3 Performance Evaluation

In this section, we evaluate the performance of different load-balancing algorithms. For this evaluation, we implemented the following load-balancing algorithms: Random Choice (RC); Round-Robin (RR); Least Connection (LC); and our proposed Fuzzy Decision (FD) algorithm. In Figure 16, when the current number of connections in the experimental desktop-as-a-service system increased from zero to 1400 connections, the average response time for a new client also increased; however, when we used different load-balancing algorithms, our fuzzy decision algorithm had shorter response times than all other algorithms, indicating that if the load-balancing algorithm can evenly dispatch client requests to terminal services according to current load information, the entire system can indeed achieve higher performance.

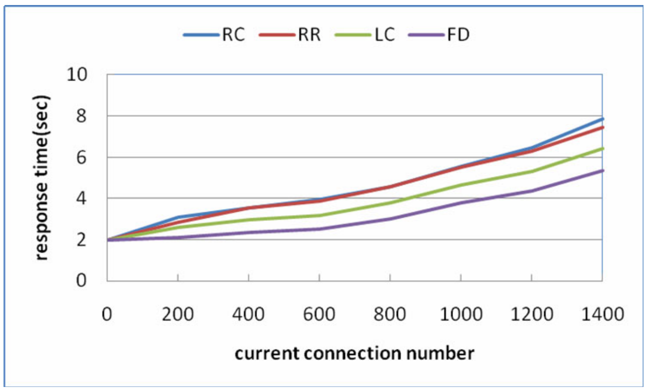


Figure 16. Performance evaluations for the four comparative algorithms

7 Conclusions

In this paper, we proposed a desktop-as-a-service system that can provide a platform for students. This system is consisting of several terminal services, and to efficiently share the load across all terminal services, we developed a load-balancing mechanism. There are several existing load-balancing algorithms, but none of them evenly dispatches client requests to the most appropriate terminal service. We showed that our proposed fuzzy decision algorithm is able to solve this issue.

Our fuzzy decision algorithm uses fuzzification, rule evaluation, and defuzzification as the three key steps to obtaining a final crisp value for each terminal service. The terminal service with the highest crisp value is then deemed the most appropriate one to serve the current client's request. Our experimental results show that our proposed fuzzy decision algorithm is able to achieve the lowest response times in comparison with other load-balancing algorithms.

References

[1] Microsoft, Microsoft Support, *Remote Desktop Protocol settings in Windows Server 2003 and in Windows XP*, http://support.microsoft.com/kb/885187.

[2] Microsoft, Windows Server, *Remote Desktop Services Forum*, http://social.technet.microsoft.com/Forums/en-US/winserverTS/threads.

[3] V. Sreenivas, M. Prathap, M. Kemal, Load Balancing Techniques: Major Challenge in Cloud Computing- A Systematic Review, *Proc. of International Conference on Electronics and Communication Systems*, Coimbatore, India, 2014, pp. 1-6.

[4] B. Addis, D. Ardagna, B. Panicucci, M. S. Squillante, L. Zhang, A Hierarchical Approach for the Resource Management of Very Large Cloud Platforms, *IEEE Transactions on Dependable and Secure Computing*, Vol. 10, No. 5, pp. 253-272, September-October, 2013.

[5] J. Luo, L. Rao, X. Liu, Temporal Load Balancing with Service Delay Guarantees for Data Center Energy Cost Optimization, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 25, No. 3, pp. 775-784, March, 2014.

[6] C. Assi, S. Ayoubi, S. Sebbah, K. Shaban, Towards Scalable Traffic Management in Cloud Data Centers, *IEEE Transactions on Communications*, Vol. 62, No. 3, pp. 1033-1045, March, 2014.

[7] Y. Sahu, R. K. Pateriya, R. K. Gupta, Cloud Server Optimization with Load Balancing and Green Computing Techniques Using Dynamic Compare and Balance Algorithm, *Proc. of 5th International Conference on Computational Intelligence and Communication Networks*, Mathura, India, 2013, pp. 527-531.

[8] G. Xu, J. Pang, X. Fu, A Load Balancing Model Based on Cloud Partitioning for the Public Cloud, *Tsinghua Science and Technology*, Vol. 18, No. 1, pp. 34-39, February, 2013.

[9] T.-L. Pao, J.-B. Chen, Cost-effective Web Cluster Mechanism for Burst Traffic, *WSEAS Transactions on Computers*, Vol. 6, No. 4, pp. 666-673, April, 2007.

[10] H.-S. Wu, C.-J. Wang, J.-Y. Xie, TeraScaler ELB-an Algorithm of Prediction-Based Elastic Load Balancing Resource Management in Cloud Computing, *Proc. of 27th International Conference on Advanced Information Networking and Applications Workshops*, Barcelona, Spain, 2013, pp. 649-654.

[11] J.-B. Chen, Efficient Content Placement on Multimedia CDN using Fuzzy Decision Algorithm, *Applied Mathematics & Information Sciences*, Vol. 6, No. 2S, pp. 471-477, April, 2012.

[12] J.-B. Chen, Fuzzy Based Approach for P2P File Sharing Detection, *Journal of Internet Technology*, Vol. 12, No.6, pp. 921-929, November, 2011.

[13] Z.-G.Chen, H.-S. Kang, S.-R. Kim, Design of a New Efficient Hybrid System for Intrusion Detection Based on HSM Fuzzy Decision Tree, *Journal of Internet Technology*, Vol. 16, No.5, pp. 885-891, September, 2015.

[14] B. Radojevic, M. Zagar, Analysis of Issues with Load Balancing Algorithms in Hosted (Cloud) Environments, *Proc. of the 34th International Convention of MIPRO*, Opatija, Croatia, 2011, pp. 416-420.

[15] Z. Zhang, L. Xiao, Y. Tao, J. Tian, S. Wang, H. Liu, A Model Based Load-Balancing Method in IaaS Cloud, *Proc. of 42nd International Conference on Parallel Processing*, Lyon, France, 2013, pp. 808-816.

[16] E. Al-Rayis, H. Kurdi, Performance Analysis of Load Balancing Architectures in Cloud Computing, *Proc. of European Modelling Symposium*, Manchester, UK, 2013, pp. 520-524.

[17] G. Soni, M. Kalra, A Novel Approach for Load Balancing in Cloud Data Center, *Proc. of Advance Computing Conference*, Gurgaon, India, 2014, pp. 807-812.

[18] C.-C. Li, K. Wang, An SLA-aware Load Balancing Scheme for Cloud Data Centers, *Proc. of International Conference on Information Networking*, Phuket, Thailand, 2014, pp. 58-63.

Biographies

Hsin-Hung Chen received his B.S. degree in the Department of Information Management from Ming Chuan University, Taoyuan, Taiwan, Republic of China, in 2001 and M.S. degree in Computer Science and Engineering in Tatung University, Taipei, Taiwan, Republic of China, in 2009. Currently, he is a Ph.D. student in Computer Science and Engineering at Tatung University. His research interests include network management, and load balance.

Li-Shing Huang received his B.B.A. degree in Tourism from Ming Chuan University, Taoyuan, Taiwan, Republic of China, in 2002 and M.S. degree in Computer Science and Engineering in Tatung University, Taipei, Taiwan, Republic of China, in 2009. Currently, he is a Ph.D. candidate in Computer Science and Engineering at Tatung University. His research interests include cloud computing and computer network management.

Jian-Bo Chen received the M.S. degree in the department of electrical engineering in National Taiwan University, Taipei, Taiwan, Republic of China, in 1995, and PhD degree in the department of computer science and engineering in Tatung University, Taipei, Taiwan, Republic of China, in 2008. He is currently an assistant professor in the department of information and telecommunications engineering in Ming Chuan University, Taoyuan, Taiwan, Republic of China. His research interests include network management, network security, and load balance.

Tsang-Long Pao received the B.S. degree in electrical engineering from Tatung Institute of Technology, Taipei, Taiwan, Republic of China, in 1982, and the M.S. and Ph.D. degree in the School of Electrical and Computer Engineering from the Georgia Institute of Technology in 1990 and 1993, respectively. He is currently a Professor in the department of computer science and engineering at the Tatung University, Taipei, Taiwan, Republic of China. His research interests include emotional speech recognition, ultrasound transducer array, ultrasound signal processing, digital image processing, and computer network management.