Exploiting a Self-learning Predictor for Session-based Remote Management Systems in a Large-scale Environment

Kuen-Min Lee[1,2], Wei-Guang Teng[1], Mu-Kai Huang[2], Chih-Pin Freg[1], Ting-Wei Hou[1]

[1] Department of Engineering Science, National Cheng Kung University, Taiwan
[2] Information and Communications Research Laboratories,
Industrial Technology Research Institute, Taiwan
allen_lee@itri.org.tw, wgteng@mail.ncku.edu.tw, mkhuang@itri.org.tw,
cpfong@fotech.edu.tw, hou@nc.es.ncku.edu.tw

Abstract

Session-based remote management systems, e.g., customer premises equipment (CPE) WAN management protocol (CWMP), have predictable task counts in a session and each CPE only accesses its own data. When the systems are used in large-scale environments, a static load balancing (LB) policy can be applied with fewer session migrations. Nevertheless, unexpected crash events, e.g., software bugs or improper management, would cause the LB policy to be reassigned so as to degrade the system performance. A self-learning predictor (SLP) is thus proposed in this work to predict unexpected crash events and to achieve a better system performance in terms of resource usage and throughput. Specifically, the SLP records and monitors all crash patterns to evaluate the system stability. Moreover, the relation flags and probabilities of all crash patterns are dynamically updated for quick comparisons. If the SLP finds the current pattern is similar to a crash pattern, a migration request is raised to the load balancer to prevent performance degradation caused by the incoming crash. The simulation results indicate that a better system performance is obtained in a large-scale environment with the proposed SLP, especially as the number of servers in each cluster node increases.

Keywords: Session-based, CWMP, Self-learning, Predictor, Crash pattern

1 Introduction

The increasing number of on-demand applications has spurred the demand for load balancing (LB) because a single server may be overloaded by a large number of simultaneous requests. On the other hand, system overloading can cause service degradations that may result in a significant loss in business credits. Thus, LBs are introduced in large-scale environments to distribute workloads across multiple computing resources and to optimize resource usage, throughput, and response time, as well as to prevent systems from overloading [1-3]. In addition, the concept of LB has been widely applied in various areas, e.g., network applications [4-5], file systems [6-7], cloud computing environments [8-11], data centers [12-14], and mobile sensor networks [15-19]. In general, LBs can be classified into two categories, i.e., *static* and *dynamic* algorithms [20]. In static LB algorithms [21], properties and capabilities of the system need to be acquired in advance, and should not vary with different system status. Thus, the execution of the LB is decided before the assignment of the tasks to the servers. On the contrary, dynamic LB algorithms are based on the current system status and redistribute tasks among individual servers during execution time [22]. A dynamic LB can be either *centralized* or *distributed*. Specifically, a single server is considered as a central node in the network to be responsible for all load distribution in centralized load distributions, while the responsibility is divided among all servers equally in distributed ones.

Another typical application of using LB is web browsing, during which much of the user behavior is unpredictable. LBs thus require monitoring parameters of servers and networks to optimize performance. However, conventional LBs do not consider unexpected crash events. This is because many conventional LBs adopt the threshold mechanism, i.e., when the load of the system and/or network reaches a predefined threshold, a request for LB is raised. The unexpected crash events cannot be monitored and managed via the threshold mechanism. Note that crash events caused by software bugs or improper management may also occur repetitively. For instance, a JavaServer Pages (JSP) web designer mistakenly applies the "System.out.println" function to log an enormous amount of information. This could cause the I/O queue to be full of exceptions. Once the log files deplete the available space within the system volume, the web server then crashes. An additional example is

*Corresponding Author: Wei-Guang Teng; E-mail: wgteng@mail.ncku.edu.tw
DOI: 10.3966/160792642018051903004

the launch of an improper or incorrect driver, especially an I/O driver. This may result in a kernel error that leads to a crash.

The cases of LB in session-based remote management systems are quite different. For example, in the customer premises equipment (CPE) WAN management protocol (CWMP) [23], all communications and operations between CPEs and an auto configuration server (ACS) are invoked with the establishment of sessions. The capabilities of servers and networks are also predefined and prearranged prior to the setup of the management system. The metrics in session-based remote management systems, e.g., the maximum number of sessions, the number of tasks, the start time, the end time, the duration, and workloads, are known and predefined by the management system. Therefore, LBs in session-based remote management systems are straightforward and efficient with static arrangements due to fewer session migrations. Consequently, predicting crash events and preventing the reassignment of LB are the major tasks of the managements used in a large-scale environment because, once an unexpected crash event occurs, the LB policy has to be reassigned from the first static arrangement.

Therefore, the repeatability feature of unexpected crash events is considered in this research, and a self-learning predictor (SLP) is proposed to predict unexpected crash events and enables the system performance of session-based remote managements in an efficient way. First, the SLP dynamically obtains the patterns from each status queue of each server and aggregates the patterns into a single blending pattern. The SLP subsequently evaluates the similarity between the current pattern and historical crash patterns. If the SLP finds a match or similar pattern, a migration request is raised to the load balancer. Finally, the SLP dynamically updates the relation flags and probabilities of all crash patterns to achieve self-learning. The relation flags are devised for quick comparisons between the blending pattern and crash patterns. The probability parameter represents the probability of each crash pattern that leads the system to crash. To sum up, the main contributions of this work are the following two points. First, a precise and efficient prediction of unexpected crash events is achieved in this study. The other one is the realization of a more effective system performance in terms of throughput and resources utilization for session-based management systems used in large-scale environments.

The rest of this paper is organized as follows. The LBs and related techniques are comprehensively reviewed in Section 2. The detailed design of the proposed SLP is in Section 3. In Section 4, the experimental setup to verify the feasibility of the proposed scheme is illustrated, and the evaluation results are also presented. Finally, Section 5 concludes the study.

2 Related Works

When the management system is used in a large-scale environment, LB is the most common technique to be employed to enable the system performance in an efficient way. A number of prior works introduce static LBs in many research areas [24-25]. For example, some DNS (domain name system) oriented LBs are used to distribute workloads among a cluster of machines [24]. Also, the parameters that contribute to the current load of the system are considered in numerous hardware-based LB solutions to optimize the system performance [25].

Similar to static LBs, dynamic strategies have also been widely adopted in the last decade [26-28]. For instance, a service-aware adaptive link load balancing mechanism is conducted to balance workloads throughout networks [26]. Many dynamic LBs are also proposed to be used with a web server queueing algorithm [27]. Several types of dynamic LBs using distributed mechanisms have been the subject of many research works, e.g., a LB is pointed out that propagates the load information about the underloaded processors in the system to the overloaded processors, and makes probabilistic transfer of work units to obtain a good load distribution [28].

On the other hand, much attention has been directed toward LBs applied in session-based remote management systems, e.g., the CWMP [29-33]. Several issues and challenges of the resource management in a remote small cell network managed by the CWMP have been discussed and solved [29]. A power traffic sharing algorithm has also been implemented for the LB in small cell networks [30]. Nevertheless, the LB cannot be directly applied in the CWMP. Therefore, many experiments of static LBs have been proposed regarding the system performance enhancements in the CWMP. For example, a hybrid static LB has been carried out in the CWMP to improve the overall system performance [31]. Another static LB with a dynamic distribution mechanism is also reported to improve the system performance of the CWMP [32]. In these two studies [31-32], static LBs are widely applied in the CWMP. This is because the capabilities of servers and networks are predefined and prearranged prior to the setup of the management system in the CWMP. Therefore, LBs in session-based remote management systems are straightforward and efficient with static arrangements because many session migrations can be avoided. Conversely, dynamic LBs are not suitable for session-based remote management systems. The major drawback of dynamic LBs is the run-time overhead due to the time consumption for selection of processes, processing for job transformations, and the communication delay due to task relocation itself [1]. Furthermore, since dynamic algorithms should collect and react to system states, more complicated than static algorithms are essentially in the CWMP [33].

Consequently, preventing reassignments of LBs are the major jobs of session-based remote management systems used in large-scale environments, e.g., caused by unexpected crash events.

Several efforts have thus been made to achieve a more effective LB solution with prediction algorithms [33-36]. For instance, a dynamic LB has been discussed to predict the future loading of nodes based on the statistics of seasonal changes [33]. Another LB adopts the historical load data to predict the future load level, and transform the prediction into the traditional classification problem [34]. An efficient LB with load prediction is achieved to optimize the performance in terms of server resource utilization with minimum energy consumption [35]. A workload prediction method using grey forecasting model is also introduced to predict the workload [36]. Since the prediction of unexpected crash events that affect the system performance in a large-scale environment is not mentioned and considered in previous works, they are thus not suitable to be applied to session-based remote management systems in large-scale environments. Therefore, an efficient and precise prediction for unexpected crash events is needed to be developed for the management systems used in large-scale environments, and the parameters of server capabilities and access objects must be considered simultaneously.

To sum up, the comparisons between the proposed SLP that combines with a static LB and other previous mechanisms have been summarized in Table 1.

Table 1. Comparisons between the proposed SLP that combines with a static LB and other previous mechanisms

Mechanisms	LB policies	Advantages	Disadvantages
Janbeglou [24]	Static LB with round-robin	DNS-based security approach is achieved	DNS protocol is needed and limited to network applications
Li [25]	Static LB with weighted least-connections	LB and high availability are provided with an efficient rescheduling mechanism	High cost with hardware-based solution
Shang [26]	Dynamic LB with centralized	Path of the best link quality is dynamically selected to meets the Qos constraint	Optimization strategy results in much time consumption
Harikesh [27]	Dynamic LB with distributed	Drop rates are minimized in homogeneous environments	More suitable for high traffic case of web servers only
Harshitha [28]	Dynamic LB with centralized	Effective load distribution while incurring less overhead	Randomized algorithm may be improved in advance
Chuang [29]	Dynamic LB with distributed	Efficient resource management with LB strategy	Focus on application in small cell networks
Fortes [30]	Dynamic LB with centralized	Proposed power traffic sharing algorithm is proper for mobile environments	Application is limited to mobile environments
Lee [31]	Static LB with hybrid sticky and replication	Suitable for environments with failure rates from low to high	A threshold that affects performance needs to be predefined manually
Lee [32]	Static LB with replication	Effective resource utilization with LB	None prediction for unexpected crash events is exploited
Sarika [33]	Dynamic LB with distributed	More data transfer with minimum response time	Prediction depends on statistics of seasonal changes may not precise enough
Tong [34]	Dynamic LB with distributed	Prediction problem is simplified into traditional classification transformation	Numerous data is required for prediction and classification methods and vary depend on different applications
Nagpure [35]	Dynamic LB with distributed	Efficient LB with future load prediction	Prediction depends on processor and memory utilization, may also consider network connections and access objects
Jheng [36]	Dynamic LB with distributed	Fewer data is applied to predict workloads accurately	Poor and unstable performance in fluctuant conditions
Proposed SLP	Static LB with the proposed SLP	Many migrations and crashes are avoided to improve throughput and resources utilization with a precise prediction	Extra predictor is essential for unexpected crash events predictions

3 System Design

3.1 System Architecture

Figure 1 shows the proposed architecture in a large-scale environment. The incoming requests issued by clients are directed to a load balancer. In the CWMP environment, a static LB policy is applied. As a result, which cluster node should handle the requests is prearranged. The load balancer then forwards the requests to the predefined cluster node.

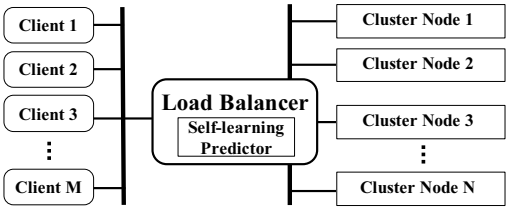

Figure 1. Architecture in a large-scale environment

The proposed SLP is created in the load balancer for session-based remote management systems to continually monitor hardware parameters, network statuses, and crash events to evaluate and predict the stability of the system in a large-scale environment. Figure 2 displays the proposed system architecture of the SLP. The Hardware Performance Monitor is responsible for monitoring the status of all the servers, including CPU usage, CPU temperature, memory usage, storage usage, and other hardware parameters. The Network Status Monitor monitors the network connection time, network connection count, and status of the networks. The Blending Module subsequently normalizes the output patterns from the above two components and aggregates the output patterns into a single blending pattern. Note that the Crash Monitor monitors all access objects and records the crash patterns in the Crash Pattern Table. The key component, the Crash Prediction Module, is in charge of acquiring the outputs from the Blending Module and Crash Pattern Table, and then dynamically evaluates the stability of the system by predicting if the current pattern will subsequently lead the system into an unstable state. Finally, depending on the prediction, the Evaluation Module is invoked to adjust and update the probability and relation flags of each crash pattern, which are stored in the Crash Pattern Table.

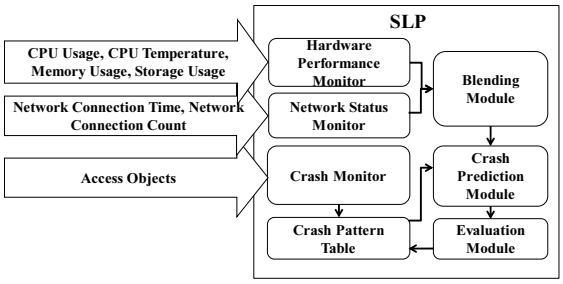

Figure 2. System architecture of the proposed SLP

3.2. System Flows

The SLP is divided into the Blending Phase (BP), Checking Phase (CP), Detecting Phase (DP), and Estimating Phase (EP). Related notations and their descriptions are summarized in Table 2 to facilitate the description of the system flows. The detailed system flows of the SLP, from Step 1 to Step 10, are illustrated in Figure 3. Each server has its own fixed-length status queue. Object access records, CPU usage, CPU temperature, memory usage, storage usage, network connection time, and connection count are all stored in each server status queue.

Table 2. Notations for the proposed SLP

Variable	Detailed Descriptions
N_e	Pattern counts, including CPU, memory, and connection.
Q_{type}	Status queue set, including Q_{cpu}, Q_{mem}, and Q_{conn}.
L_q	Status queue length
P	Pattern set $P=\{P_{type}\}$, including P_{cpu}, P_{mem}, and P_{conn}.
L_p	Pattern length
R_{type}	Relation flag of each corresponding P_{type}
W_{type}	Weighted value of each corresponding P_{type}
s	Sub-pattern length, where sub-pattern means the original pattern has been blended with shorter length.
d	Weighted Manhattan distance
p'	Corresponding sub-pattern in Crash Pattern Table
P_{max}	Predefined maximum distance used in the weighted Manhattan distance approach
$Prob$	Probability of each pattern leads the system to crash
c	Number of crash events; initial value c=1
m	Number of mispredictions; initial value m=0
a	Number of accurate predictions; initial value a=0
α	Decreasing factor of $Prob$
β	Increasing factor of $Prob$
T_S	Threshold of similarity between objects
SP	Safe point in the weighted Manhattan distance approach
T_R	Threshold of the distance to identify the dependence of relationships

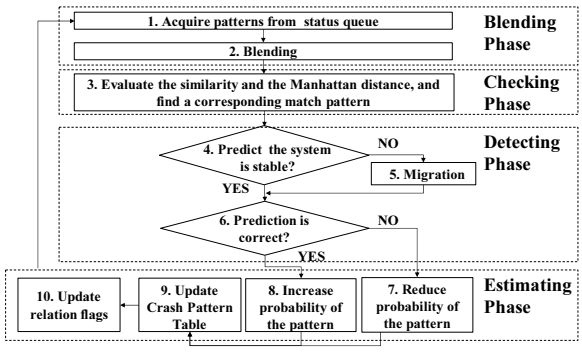

Figure 3. Flows of the proposed SLP

The first phase of the SLP, the Blending Phase, acquires patterns from each server's status queue (Step 1) and aggregates the patterns into a single blending

pattern (Step 2). The Blending Phase also builds up the relation flags for each pattern and records them in the Crash Pattern Table. The relation flags are implemented to quickly evaluate the stability of the system by ignoring unrelated patterns. To have an efficient and fast evaluation, the BP also extracts the metrics in the status queue, e.g., CPU usage, memory usage, and connection count, to form a sub-pattern with the length of $s = L_p / N_e$. The content of ith parameter in sub-patterns of set P, which is defined as pattern set $\{P_{type}\}$, is denoted as in Eq. (1). Note that the objective of Eq. (1) is to extract the original pattern into a blending pattern with shorter length to save computing resource. Besides, $Q_{type(j)}$ indicates each element j in the status queue set Q_{type}, including Q_{cpu}, Q_{mem}, and Q_{conn}.

$$P_{type}[i] = \frac{s}{L_q}\left(\sum_{j=i\times\frac{L_q}{s}}^{\frac{L_q}{s}(i+1)-1} Q_{type(j)} \right) \quad \textbf{(1)}$$

The second phase of the SLP, the Checking Phase, regularly checks whether the current blending pattern is similar to an identified crash pattern in the Crash Pattern Table (Step 3). A weighted Manhattan distance approach [37] is applied to evaluate the similarity and find a corresponding one if the similarity is larger than a predefined threshold T_S. For instance, the parameter of the CPU usage in the current blending pattern only needs to be compared with the CPU usage in the crash pattern, and it does not need to be compared with other parameters in the crash pattern. In addition, for the Manhattan distance approach, the distance between two points in a grid based on a strictly horizontal and/or vertical path, the Manhattan distance is the simple sum of the horizontal and vertical components. The Manhattan distance approach is thus a simple and time-saving comparison approach. Moreover, weighted values are added to the Manhattan distance approach to achieve an efficient prediction of the stability of the system. The weighted value W_{type} in the weighted Manhattan distance approach is represented as in Eq. (2).

$$W_{type} = R_{type} / \sum_{each\ \ type\ \ i} R_i \quad \textbf{(2)}$$

The expression of weighted Manhattan distance d in the weighted Manhattan distance approach is defined as in Eq. (3). Note that p'_j is the corresponding sub-pattern of the original pattern p_j in Crash Pattern Table. Furthermore, a predefined maximum distance P_{max} is used to obtain the Manhattan distance, and the weighted value of each type w_i, which is defined as W_{type} in Eq. (2), is considered to calculate the final weighted Manhattan distance d of each blending pattern.

$$d = \sum_{eachtype\ \ i} w_i \left(\sum_{each\ \ j\ \ in\ \ p_i} p_{\max} - \left| p'_j - p_j \right| \right) \quad \textbf{(3)}$$

If the CP finds a match or a similar pattern, the SLP subsequently enters the third phase, the Detecting Phase, to evaluate and predict the stability of the system (Step 4). If the DP predicts that the current blending pattern is a crash event, a migration is triggered (Step 5). The DP then checks whether the prediction is correct for the EP (Step 6).

Finally, the Estimating Phase updates the probability and relation flags of each crash pattern in the Crash Pattern Table according to the predictions. The probability of each pattern leading to a system crash, *Prob*, is defined by heuristic rules as indicated in Eq. (4). In the design, the increasing and decreasing factors update slowly in the first several rounds. The factors subsequently update at an exponential growth rate and also update slowly in the final saturated state.

$$\text{Prob} = \frac{1}{e^{\alpha}} \times \ln(c+1) \quad \textbf{(4)}$$

If the DP made a wrong prediction (Step 7), the EP updates the variable m as $m + 1$, which means the number of mispredictions increases by 1. The probability *Prob* is decreased as in Eq. (5).

$$\text{Prob} = \max\left(0, \text{Prob} - \frac{\ln(\text{m})}{\text{e}^{\alpha}}\right) \quad \textbf{(5)}$$

On the other hand, if the prediction of the DP is correct (Step 8), the EP subsequently increases the variable a to $a + 1$. The probability *Prob* is increased as in Eq. (6).

$$\text{Prob} = \min\left(1, \text{Prob} + \frac{\ln(\text{a})}{\text{e}^{\beta}}\right) \quad \textbf{(6)}$$

Finally, the related contents of the Crash Pattern Table are updated by the EP (Step 9). The relation flags of these changed patterns are also updated by the EP (Step 10). After the EP is completed, the system flow goes back to the BP, and the unexpected crash events are monitored and predicted by both the CP and DP.

4 Experimental Results and Discussion

Several experimental simulations were performed to demonstrate the feasibility of the proposed SLP, including the prediction precision and the overall system performance in a large-scale environment. The parameters used in the proposed experimental environments to evidence the average precision and prediction time are explored in Section 4.1. Evaluation results in terms of memory usage, time consumption, and connection failure are shown in Section 4.2 to verify the performance of the proposed SLP combines with a static LB policy.

4.1 Evaluation of the Prediction Precision

To depict the feasibility of the SLP, the test strategy is to verify the prediction precision of the SLP with various conditions and situations, e.g., with considering different memory requirements and different numbers of execution loops. Therefore, the experiments in this study are verified with two different tasks using the CWMP protocol. The first task, Type A, has a low memory requirement (20 to 40 MB with normal distribution). The other task, Type B, has a high memory requirement (80 to 110 MB with inverse normal distribution). Note that the tails of the inverse normal distribution decrease more slowly than the normal distribution, and are thus more suitable to model numerically large values. In addition, the number of tasks for Type A and Type B is set to 1,200. Table 3 summarizes the experimental setup values of the variables for the proposed SLP.

Table 3. Experimental setup values of the variables for the proposed SLP

Variable	Experimental Setup Value
L_q	100
s	3
α	4
β	2
$Prob$	0.8
L_p	5
T_S	40
SP	30
P_{max}	100
T_R	10

In the CWMP implementation, the execution period of the remote procedure call (RPC) is set as 60 seconds. Moreover, 30 specific rules are predefined to denote the crash events. For instance, when the access object ID 99 under the CPU usage is more than 90% and the memory usage is greater than 80%, the system subsequently crashes. An additional example of the crash rule is when access to object ID 55 behind object ID 90 under the capacity of the network connection is greater than 95%. In the simulation, all variables including object access records, CPU usage, CPU temperature, memory usage, storage usage, network connection time, and connection count are generated randomly. When the generated pattern matches the predefined rules, the pattern is identified as a crash event that leads to a system crash. As the crash rules are predefined, all crash events belong to the specific types of patterns. This means the same types of crash events occur repetitively. As a result, the repeatability feature of unexpected crash events is simulated. The test bed of the ACS server was equipped with Intel Core i7 920 2.66 GHz, DDR3-2133 8 GB, Windows 7 Professional 64-bits, and JavaSE-1.7 environment. All the experimental results were simulated more than 500 times to demonstrate the feasibility of the SLP.

Average precision. Two cases with 20 rounds, including different memory requirements and different numbers of execution loops, were simulated. Note that the definition of each execution loop indicates that the ACS completes all tasks requested by the CPEs. In the case of different memory requirements, the memory requirement of Type B is set between 80 and 110 MB deliberately, and the number of execution loops is set from 1,000 to 11,000 times. As indicated in Figure 4, the SLP spends a period of time collecting crash patterns and recording them in the Crash Pattern Table. During this time, there are few crash patterns in the Crash Pattern Table to be compared. Few predictions are made, and the true prediction (precision) in the first round is 71.4%. Specifically, the definition of precision is the SLP correctly predicting the crash events over all predictions, including true predictions and false predictions. Nevertheless, it indicates that the SLP prediction has a high hit rate in the initial state. After the second round, the precision is up to 90.5%, which indicates that the SLP prediction is both efficient and precise. Additionally, the average precision is greater than 93% after 5 rounds. The precision thus becomes steady after only a few rounds because there tends to be specific types of unexpected crash events, and the self-learning mechanism does an efficient and precise prediction in the following rounds.

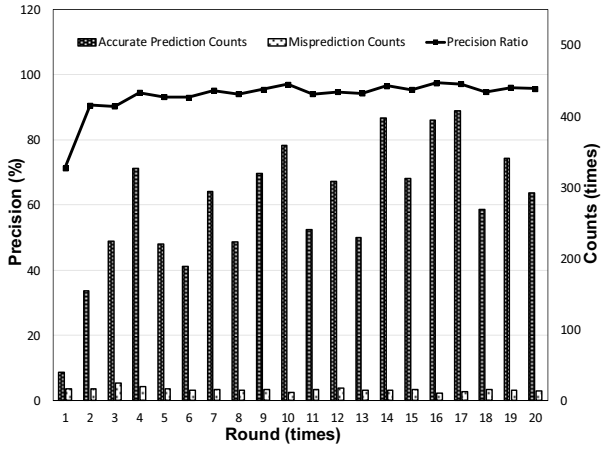

Figure 4. Round times vs. average precision given different memory requirements

In the other case, different numbers of execution loops are considered. The number of execution loops of Type A is set between 8,000 and 11,000 times deliberately, and the memory requirement is set between 20 and 40 MB. As indicated in Figure 5, the results demonstrate that the precision values in the first and the second rounds are 74.2% and 92.5%, respectively. It also proves the high hit rate for the SLP to predict crash events. Few predictions are in the first round because only several crash patterns in the Crash Pattern Table are compared for the SLP. After 5 rounds, the average precision is up to 93%. This also indicates that the precision becomes steady after only a few rounds because of the repeatability feature of

unexpected crash events. One may readily observe that the self-learning mechanism makes effective and precise predictions for the CWMP network.

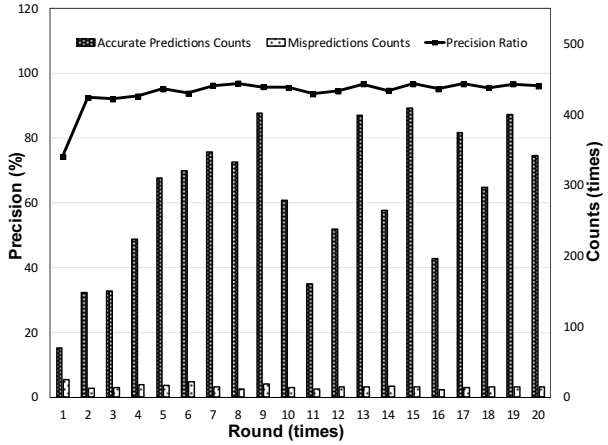

Figure 5. Round times vs. average precision given different numbers of execution loops

Average prediction time. Table 4 represents two simulated cases of the average prediction time within 5 rounds using the CWMP as an example. These two cases include the crash case and the non-crash case. As indicated by the experimental results, the first round in the crash case takes the most time, approximately 0.34 ms, because the SLP spends a considerable amount of time recording all crash events in the Crash Pattern Table. However, the prediction time of the other rounds are all under 0.16 ms, and the average prediction time is 0.1 ms. As the execution period of the RPC is set to 60 seconds, the average prediction time is thus less than 0.00056% of the execution period even in the case of the largest required time (0.34 ms). Consequently, the results indicate that the proposed SLP is a precise and time-saving prediction mechanism.

Table 4. Average prediction time in different cases within 5 rounds

	Average Prediction Time: Crash Case (msec.)	Average Prediction Time: Non-Crash Case (msec.)
Round 1	0.333436	0.009727
Round 2	0.141699	0.011907
Round 3	0.153717	0.011812
Round 4	0.154657	0.016890
Round 5	0.155137	0.009998

4.2 Evaluation of the Overall System Performance

The main objective of a LB used in a large-scale environment is to maximize the resource usage and system throughput. Hence, the test strategy to verify the overall system performance is to compare the resource utilization (e.g., memory usage) and the throughput (e.g., the execution time while handling the same number of tasks or sessions) among various LB mechanisms. On the other hand, to evaluate the system performance when used with the proposed SLP, a simple static LB, i.e., a sticky session management is applied to combine with the SLP. In the sticky session management, once a session is established, it is sent to one of the cluster servers based on a decision made by the load balancer. Then, all subsequent requests in this session are directed to the same server. The major advantage of this technique is that the memory space is well utilized. Conversely, once a session is established in the session replication, it is replicated to all cluster servers. The major advantage of this technique is its rapid recovery from unexpected crash events (e.g., due to network failure or a down server).

Six different techniques are compared in the experiments, including the proposed SLP combines with a sticky session, the sticky session, the session replication, the primary-secondary (PS) session replication in J2EE session management [38], a hybrid technique [31], and a dynamic distribution (DD) technique [32]. Note that the PS scheme always duplicates sessions to only one or two cluster nodes, whereas the hybrid technique duplicates sessions to all servers once the connection failure rate reaches a predefined threshold, and it adopts the sticky session if the failure rate is low. The number of cluster nodes to be replicated in the DD technique depends on the network situation and the connection failure rates.

As mentioned in a session admission control mechanism in a previous work [39], a high connection failure rate results from the situations when clients are unable to get the responses from a server (e.g., because of unstable wireless links or overloaded servers). Therefore, a wide range of high connection failure rate that leads the system to crash, up to 10%, is used to simulate the target environment. The simulations are continuously performed as the crash rate grows from 1% to 10%. An environment including diverse clients with wired or wireless connections is simulated to depict a wide range of crash rates by developing software programs. Specifically, in the target environment, the connections among servers in a cluster node are generally wired links whereas clients may connect to a server cluster with either wired or wireless links. Additionally, according to a previous report [40], most resources are occupied by resource-intensive tasks that have long durations. The memory demands of a memory-intensive task could be greater than 1 gigabyte. Hence, redundant memory usage should be reduced for session management, especially in a large-scale environment and considering the overall system performance. Consequently, the experimental environments are set as shown in Table 5.

Table 5. Parameters used in the simulation of overall system performance evaluation

Parameter	Value
Number of clients	10,000
Number of cluster nodes	5-10
Number of sessions	1,000,000
Crash rate	1%-10%

In the simulation, the network architecture of simulations is illustrated as shown in Figure 1. In addition, an Exponential distribution is applied to create the time consumption of the session operations, such as the time of session connection, session creation, and session replication. The mean of the Exponential distribution is set at 100 ms for both the connection time and session replication. For the session creation time, the mean of the Exponential distribution is 1,000 ms. Similarly, in the memory usage simulation, the consumption of memory space of each session is generated by Gaussian distribution with an average of 200 bytes and a standard deviation of 50 bytes. Note that the self-learning mechanism in the SLP is continuously adopted to learn and predict unexpected crash events as the crash rate increases from 1% to 10%. This means the predicted results in previous simulations are also applied to predict unexpected crash events in the next simulation.

Memory usage. Figure 6 and Figure 7 show the comparison of total memory usage of all cluster nodes with server counts of 5 and 10, respectively. The data in these two figures show that the memory usage of the session replication remains constant as the crash rate increases from 1% to 10% because session replication always sends a session to all cluster nodes. On the other hand, the memory usage of the sticky session and the PS are proportional to the crash rate because when any session disconnection occurs, the system has to rebuild the session, which occupies more memory space. For the hybrid session management, the memory usage is near that of the sticky session for the case in which the crash rate is low. However, the memory usage of the hybrid and the DD techniques grow dramatically as the crash rate increases because once the crash rate increases, session replication is applied to all new sessions.

As shown in Figure 6 and Figure 7, the memory usage of the session replication, the sticky session, the hybrid, and the DD techniques greatly increase as the number of servers in each cluster node increases from 5 to 10 servers. Conversely, the memory usage of the SLP only slightly increases. Moreover, the memory usage of the SLP is comparable similar to that of the PS scheme, which is the best one among these six techniques. This is because the PS scheme always duplicates sessions to only one or two server nodes, so that less memory usage is required.

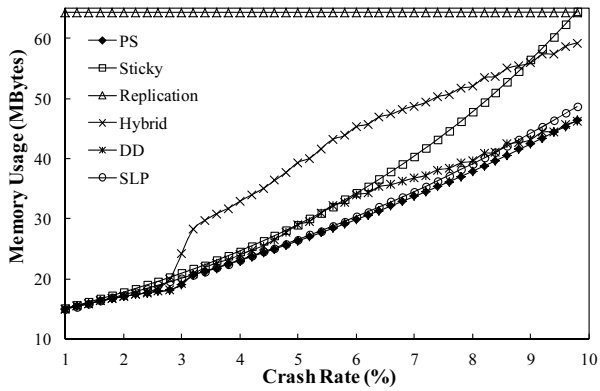

Figure 6. Comparison of memory usage with the cluster node of 5 servers

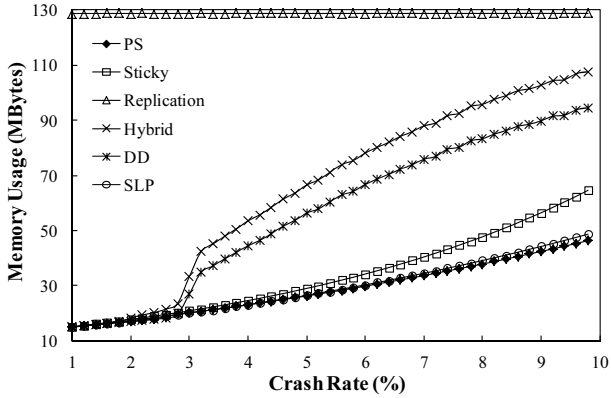

Figure 7. Comparison of memory usage with the cluster node of 10 servers

Time consumption. The comparisons of time consumption in a server cluster of 5 servers and 10 servers are demonstrated in Figure 8 and Figure 9, respectively. The results in these two figures show that the time consumption of the sticky session and PS scheme increase dramatically when the crash rate is greater than 3%. However, the time consumption of using the SLP is significantly lower than that of the sticky session and that of the PS scheme. In addition, the time consumption of the session replication is slightly greater than that of both the DD and the SLP because the session replication replicates sessions to all cluster nodes, which leads to fewer crash events.

Specifically, the time consumption of the SLP is close to that of the session replication, which is the superior one among all techniques. The main reason is that sessions are replicated to all cluster nodes in the session replication which results in fewer crash events, so that a better performance is obtained.

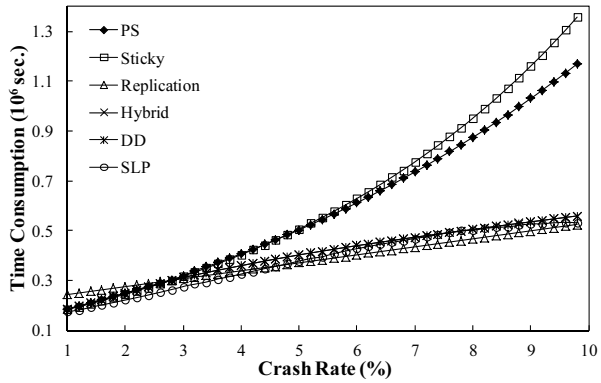

Figure 8. Comparison of time consumption with the cluster node of 5 servers

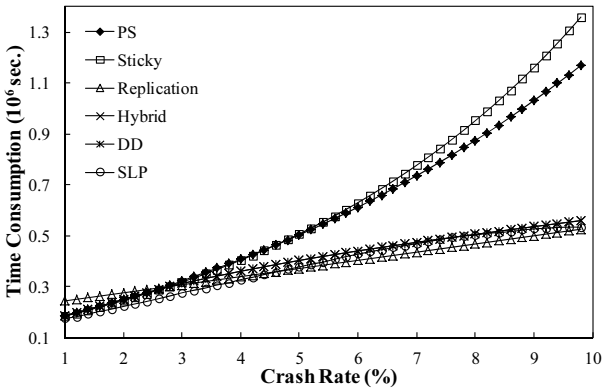

Figure 9. Comparison of time consumption with the cluster node of 10 servers

Memory time product. The evaluation measure, memory time product (MTP) is applied to provide a general performance evaluation of these techniques. MTP is the memory usage times the time consumption. Thus, a lower MTP indicates a better performance. Figure 10 and Figure 11 depict the MTPs of the SLP, the DD, the sticky session, the session replication, the PS scheme, and the hybrid technique. The results in Figure 10 and Figure 11 confirm that the static LB policy combined with the SLP has the best MTP than the other five techniques. Moreover, in Figure 10, the MTP of the SLP is slightly better than that of the DD. Nevertheless, as the number of servers in each cluster node increases from 5 to 10, the performance of the SLP is much better than that of the DD and the other techniques, as shown in Figure 11. As a result, the comparison of the MTPs shows that the SLP has the lowest MTP, meaning that the SLP generally outperforms the other techniques, especially as the number of servers in each cluster node increases.

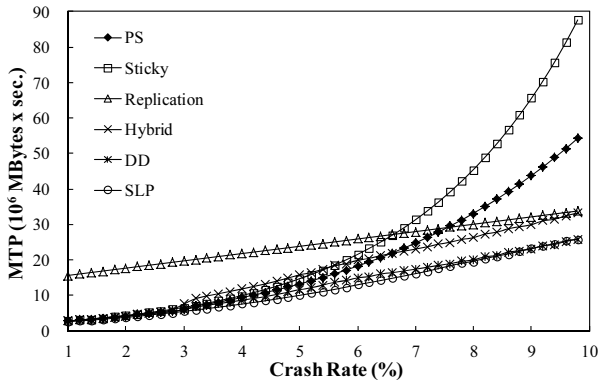

Figure 10. Comparison of MTP with the cluster node of 5 servers

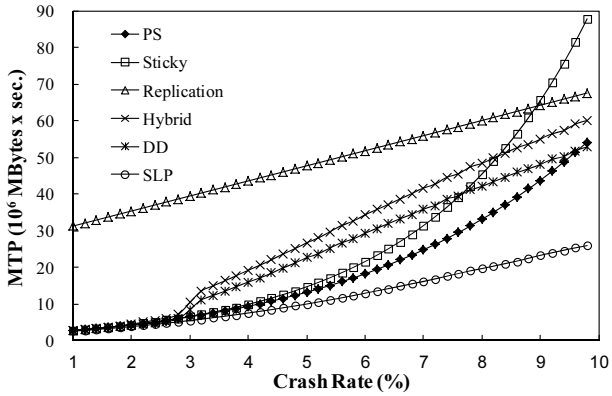

Figure 11. Comparison of MTP with the cluster node of 10 servers

Connection failure. Figure 12 depicts the comparison of the number of actual connection failures. The experiments were executed on a system with 10 servers in each cluster node. As shown in Figure 12, the failure count of the sticky session grows dramatically when the crash rate is higher than 2.8%. Once a crash occurs, the system repeatedly re-connects and re-sends until the communication is completed. A higher crash rate thus leads to frequent failures. Compared with the sticky session, the actual failure count of the SLP is much lower than that of the sticky session because a large number of crash events are predicted precisely with the self-learning mechanism in the SLP. Moreover, the actual connection failure count of the PS scheme is much higher than that of the SLP. The actual connection failure count when using the SLP is slightly greater than that when using the session replication. This finding indicates that the session replication technique outperforms the SLP in merely reducing limited crash failures, but at the cost of much higher memory consumption. In summary, the SLP eliminates most migrations that are caused by unexpected crash events with less memory usage because of highly precise predictions.

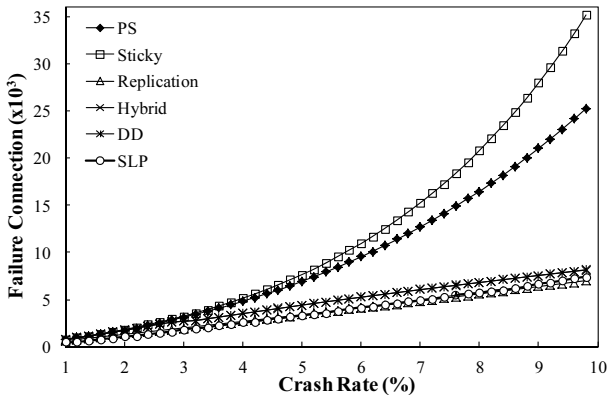

Figure 12. Comparison of actual connection failure

5 Conclusions

Unlike many LB policies developed for general applications, the predictability of metrics in a session-based remote management system have been additionally exploited, e.g., the CWMP. Furthermore, unexpected crash events are the key factors affecting the system performance of a session-based remote management system in large-scale environments. Given that crash events may occur repetitively, a self-learning predictor named SLP is proposed in this work. To verify its feasibility, the SLP that combines with a sticky session has been applied to the CWMP in a large-scale environment. From the experimental studies, the prediction of the SLP achieves a high hit rate (over 71%) in the initial state, and the precision becomes steady with a very high hit rate (over 93%) after only a few rounds (within 5 rounds). The average prediction time is equal to 0.1 ms, which is less than 0.00056% of the execution period, even in the case of the largest required time (0.34 ms). Moreover, the MTP experimental results show a static LB combined with the proposed SLP is more effective in terms of resources utilization and throughput, i.e., has a more efficient performance in terms of memory usages and the execution time while handling a large number of sessions. Consequently, with the proposed SLP, a session-based remote management system can achieve a better system performance in a large-scale environment, especially as the number of servers in each cluster node increases.

Acknowledgements

The authors would like to acknowledge Kuo-Ming Huang, Jin-Neng Wu, and Ping-Yu Chen in Industrial Technology Research Institute (ITRI) for their valuable discussions. Additionally, this work was supported in part by the Headquarters of University Advancement at the National Cheng Kung University, which is sponsored by the Ministry of Education, Taiwan.

References

[1] A. Sharma, S. Verma, A Survey Report on Load Balancing Algorithm in Grid Computing Environment, *International Journal of Advanced Engineering Research and Studies*, Vol. 4, No. 2, pp. 128-132, January-March, 2015.

[2] S. Pandey, S. Prasanna, S. Kapil, R. B. S., Load Balancing Techniques: A Comprehensive Study, *International Journal of Advanced Research in Computer Science and Management Studies*, Vol. 3, No. 4, pp. 331-335, April, 2015.

[3] M. Li, H.-H. Yang, CEA: A Cyclic Expansion Algorithm for Data Migration in Parallel Video Servers, *Journal of Parallel and Distributed Computing*, Vol. 72, No. 7, pp. 868-879, July, 2012.

[4] A. Mishra, Network Load Balancing and Its Performance Measures, *International Journal of Computer Science Trends and Technology*, Vol. 3, No. 1, pp. 77-81, January-February, 2015.

[5] C.-Y. Chang, T.-Y. Wu, C.-C. Huang, A. J.-W. Whang, H.-C. Chao, Robust Header Compression with Load Balance and Dynamic Bandwidth Aggregation Capabilities in WLAN, *Journal of Internet Technology*, Vol. 8, No. 3, pp. 365-372, July, 2007.

[6] S. C. Deshmukh, S. S. Deshmukh, A Survey: Load Balancing for Distributed File System, *International Journal of Computer Applications*, Vol. 111, No. 5, pp. 25-29, February, 2015.

[7] H.-T. Chang, Y.-M. Chang, S.-Y. Hsiao, Scalable Network File Systems with Load Balancing and Fault Tolerance for Web Services, *Journal of Systems and Software*, Vol. 93, pp. 102-109, July, 2014.

[8] N. Grozev, R. Buyya, Performance Modelling and Simulation of Three-Tier Applications in Cloud and Multi-Cloud Environments, *The Computer Journal*, Vol. 58, No. 1, pp. 1-22, January, 2015.

[9] P. D. Kaur, I. Chana, A Resource Elasticity Framework for QoS-aware Execution of Cloud Applications, *Future Generation Computer Systems*, Vol. 37, pp. 14-25, July, 2014.

[10] H. Sun, T. Zhao, Y. Tang, X. Liu, A QoS-Aware Load Balancing Policy in Multi-tenancy Environment, *Proceedings of the 8th International Symposium on Service Oriented System Engineering*, Oxford, England, 2014, pp. 140-147.

[11] M. Beltrán, Automatic Provisioning of Multi-tier Applications in Cloud Computing Environments, *The Journal of Supercomputing*, Vol. 71, No. 6, pp. 2221-2250, June, 2015.

[12] H. J. Moon, H. Hacıgümüş, Y. Chi, W.-P. Hsiung, SWAT: A Lightweight Load Balancing Method for Multitenant Databases, *Proceedings of the International Conference on Extending Database Technology*, Genoa, Italy, 2013, pp. 65-76.

[13] C.-C. Li and K. Wang, An SLA-aware Load Balancing Scheme for Cloud Datacenters, *Proceedings of the International Conference on Information Networking*, Phuket, Thailand, 2014, pp. 58-63.

[14] I. Kamel, Z. A. Aghbari, A. Mustafa, Efficient Range Queries in Spatial Databases over Peer-to-Peer Networks, *International Journal of Internet Protocol Technology*, Vol. 4, No. 2, pp. 79-90, July, 2009.

[15] Y.-S. Chen, T.-Y. Juang, Y.-W. Lin, I.-C. Tsai, A Low Propagation Delay Multi-path Routing Protocol for Underwater Sensor Networks, *Journal of Internet Technology*, Vol. 11, No. 2, pp. 153-165, March, 2010.

[16] N.-C. Wang, S.-C. Chang, A Load Balancing Data Aggregation Scheme for Grid-based Wireless Sensor Networks, *International Journal of Ad Hoc and Ubiquitous Computing*, Vol. 14, No. 4, pp. 279-287, January, 2013.

[17] N. Jain, D. K. Madathil, D. P. Agrawal, MidHopRoute: A Multiple Path Routing Framework for Load Balancing with Service Differentiation in Wireless Sensor Networks, *International Journal of Ad Hoc and Ubiquitous Computing*, Vol. 1, No. 4, pp. 210-221, July, 2006.

[18] H. Zafar, D. Harle, I. Andonovic, Y. Khawaja, Performance Evaluation of Shortest Multipath Source Routing Scheme, *IET Communications*, Vol. 3, No. 5, pp. 700-713, May, 2009.

[19] A. Jaron, P. Pangalos, A. Mihailovic, A. H. Aghvami, Proactive Autonomic Load Uniformisation with Mobility Management for Wireless Internet Protocol (IP) Access Networks, *IET Networks*, Vol. 1, No. 4, pp. 229-238, December, 2012.

[20] M. Patel, C. Jani, A Survey on Heterogeneous Load Balancing Techniques in Cloud Computing, *International Journal for Innovative Research in Science & Technology*, Vol. 1, No. 10, pp. 180-185, March, 2015.

[21] R. Kanakala, V. K. Reddy, Performance Analysis of Load Balancing Techniques in Cloud Computing Environment, *Indonesian Journal of Electrical Engineering*, Vol. 13, No. 3, pp. 568-573, March, 2015.

[22] A. A. Jaiswal, S. Jain, An Approach towards the Dynamic Load Management Techniques in Cloud Computing Environment, *Proceedings of the International Conference on Power, Automation and Communication*, Amravati, India, 2014, pp. 112-122.

[23] Broadband Forum, CPE WAN Management Protocol Amendment 5 Specification, TR-069, November, 2013.

[24] M. Janbeglou, H. Naderi, N. Brownlee, Effectiveness of DNS-based Security Approaches in Large-scale Networks, *Proceedings of the 28th International Conference on Advanced Information Networking and Applications Workshops*, Victoria, Canada, 2014, pp. 524-529.

[25] S. Li, F. Wang, B. Xiao, F. Yang, X. Sun, Y. Wang, Study of Load Balancing Technology for EAST Data Management, *Fusion Engineering and Design*, Vol. 89, No. 5, pp. 750-753, May, 2014.

[26] F. Shang, L. Mao, W. Gong, Service-aware Adaptive Link Load Balancing Mechanism for software-Defined Networking, *Future Generation Computer Systems*, Vol. 81, pp. 452-464, April, 2008.

[27] H. Singh, S. Kumar, WSQ: Web Server Queueing Algorithm for Dynamic Load Balancing, *Wireless Personal Communications*, Vol. 80, No. 1, pp. 229-245, January, 2015.

[28] H. Menon, L. Kalé, A Distributed Dynamic Load Balancer for Iterative Applications, *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Denver, CO, 2013, pp. 1-11.

[29] M.-C. Chuang, M. C. Chen, Y. S. Sun, Resource Management Issues in 5G Ultra Dense Smallcell Networks, *Proceedings of the International Conference on Information Networking*, Siem Reap, Cambodia, 2015, pp. 159-164.

[30] S. Fortes, A. Aguilar-García, R. Barco, F. B. Barba, J. A. Fernández-luque, A. Fernández-Durán, Management Architecture for Location-aware Self-organizing LTE/LTE-a small Cell Networks, *IEEE Communications Magazine*, Vol. 53, No. 1, pp. 294-302, January, 2015.

[31] K.-M. Lee, J.-N. Wu, P.-Y. Chen, Y.-C. Chao, *Remote Management System with Adaptive Session Management Mechanism*, U.S. Patent 8938495, January, 2015.

[32] K.-M. Lee, W.-G. Teng, J.-N. Wu, P.-Y. Chen, M.-K. Huang, T.-W. Hou, Dynamic Distribution Technique for Session Management in CWMP Server Clusters, *Electronics Letters*, Vol. 50, No. 22, pp. 1639-1641, October, 2014.

[33] S. V. Bodake, R. Naik, Design of Decentralized Load Balancing Algorithm for Cloud Environment, *International Advanced Research Journal in Science, Engineering and Technology*, Vol. 2, No. 5, pp. 19-24, May, 2015.

[34] J.-J. Tong, H.-H. E, M.-N. Song, J.-D. Song, Host Load Prediction in Cloud Based on Classification Methods, *The Journal of China Universities of Posts and Telecommunications*, Vol. 21, No. 4, pp. 40-46, August, 2014.

[35] M. B. Nagpure, P. Dahiwale, P. Marbate, An Efficient Dynamic Resource Allocation Strategy for VM Environment in Cloud, *Proceedings of the International Conference on Pervasive Computing*, Pune, India, 2015, pp. 1-5.

[36] J.-J. Jheng, F.-H. Tseng, H.-C. Chao, L.-D. Chou, A Novel VM Workload Prediction Using Grey Forecasting Model in Cloud Data Center, *Proceedings of the International Conference on Information Networking*, Phuket, Thailand, 2014, pp. 40-45.

[37] H. Ding, H. Wu, X. Zhang, J. Chen, Writer Identification Based on Local Contour Distribution Feature, *International Journal of Signal Processing, Image Processing and Pattern Recognition*, Vol. 7, No. 1, pp. 169-180, February, 2014.

[38] D. Rossi, E. Turrini, Analyzing the Impact of Components Replication in High Available J2EE Clusters, *Proceedings of the Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services*, Papeete, Tahiti, 2005, pp. 56-66.

[39] L. Cherkasova, P. Phaal, Session-based Admission Control: A Mechanism for Peak Load Management of Commercial Web Sites, *IEEE Transactions on Computers*, Vol. 51, No. 6, pp. 669-685, June, 2002.

[40] A. K. Mishra, J. L. Hellerstein, W. Cirne, C. R. Das, Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters, *ACM SIGMETRICS Performance Evaluation Review*, Vol. 37, No. 4, pp. 34-41, March, 2010.

Biographies

Kuen-Min Lee received a M.S. degree in the Department of Electrical Engineering, National Chung Cheng University, Chiayi, Taiwan in 1999, and received a Ph.D. degree in the Department of Engineering Science, National Cheng Kung University, Tainan, Taiwan in 2016. His research interests include home networking and cloud distributed systems.

Wei-Guang Teng received B.S. and Ph.D. degrees in the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan in 1998 and 2004, respectively. He is currently an associate professor in the Department of Engineering Science, National Cheng Kung University, Tainan, Taiwan. His research interests include data mining and multimedia networking.

Mu-Kai Huang received a M.S. degree in the Department of Electronic Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan in 2011. He is an engineer at the Information and Communications Research Laboratories, Industrial Technology Research Institute (ITRI), Tainan, Taiwan. His research interests include remote management and distributed systems.

Chih-Pin Freg received his M.S. degree in Computer Engineering from the University of Missouri at Columbia in 1990. He is currently teaching at the Fortune Institute of Technology. His major research interests include pattern recognition and mobile applications.

Ting-Wei Hou received B.S., M.S., and Ph.D. degrees in EE, National Cheng Kung University (NCKU) Tainan, Taiwan in 1983, 1985, and 1990. He joined Department of Engineering Science, NCKU, in 1990. He is now a professor and head of the department. His research interests include medical information systems and robots.