Integration of Least Recently Used Algorithm and Neuro-Fuzzy System into Client-side Web Caching

Waleed Ali prowalid_2004@yahoo.com
Faculty of Computer Science and Information System
Universiti Teknologi Malaysia (UTM)
Skudai, 81310, Johor Bahru, Malaysia

Siti Mariyam Shamsuddin mariyam@utm.my
Faculty of Computer Science and Information System
Universiti Teknologi Malaysia (UTM)
Skudai, 81310, Johor Bahru, Malaysia

ABSTRACT

Web caching is a well-known strategy for improving performance of Web-based system by keeping web objects that are likely to be used in the near future close to the client. Most of the current Web browsers still employ traditional caching policies that are not efficient in web caching. This research proposes a splitting client-side web cache to two caches, short-term cache and long-term cache. Initially, a web object is stored in short-term cache, and the web objects that are visited more than the pre-specified threshold value will be moved to long-term cache. Other objects are removed by Least Recently Used (LRU) algorithm as short-term cache is full. More significantly, when the long-term cache saturates, the neuro-fuzzy system is employed in classifying each object stored in long-term cache into either cacheable or uncacheable object. The old uncacheable objects are candidate for removing from the long-term cache. By implementing this mechanism, the cache pollution can be mitigated and the cache space can be utilized effectively. Experimental results have revealed that the proposed approach can improve the performance up to 14.8% and 17.9% in terms of hit ratio (HR) compared to LRU and Least Frequently Used (LFU). In terms of byte hit ratio (BHR), the performance is improved up to 2.57% and 26.25%, and for latency saving ratio (LSR), the performance is improved up to 8.3% and 18.9%, compared to LRU and LFU.

Keywords: Client-side web caching, Adaptive neuro-fuzzy inference system, Least Recently Used algorithm.

1. INTRODUCTION

One of the important means to improve the performance of Web service is to employ web caching mechanism. Web caching is a well-known strategy for improving performance of Web-based system. The web caching caches popular objects at location close to the clients, so it is considered one of the effective solutions to avoid Web service bottleneck, reduce traffic over the Internet and improve scalability of the Web system[1]. The web caching is implemented at client, proxy server and original server [2]. However, the client-side caching (browser caching) is

economical and effective way to improve the performance of the Word Wide Web due to the nature of browser cache that is closer to the user [3,4].

Three important issues have profound impact on caching management namely: cache algorithm (passive caching and active caching), cache replacement and cache consistency. However, the cache replacement is the core or heart of the web caching; hence, the design of efficient cache replacement algorithms is crucial for caching mechanisms achievement [5]. In general, cache replacement algorithms are also called web caching algorithms [6].

Since the apportioned space to the client-side cache is limited, the space must be utilized judiciously [3]. The term "cache pollution" means that a cache contains objects that are not frequently used in the near future. This causes a reduction of the effective cache size and affects negatively on performance of the Web caching. Even if we can locate large space for the cache, this will be not helpful since the searching for object in large cache needs long response time and extra processing overhead. Therefore, not all Web objects are equally important or preferable to store in cache. The setback in Web caching consists of what Web objects should be cached and what Web objects should be replaced to make the best use of available cache space, improve hit rates, reduce network traffic, and alleviate loads on the original server.

Most web browsers still concern traditional caching policies [3, 4] that are not efficient in web caching [6]. These policies suffer from cache pollution problem either cold cache pollution like the least recently used (LRU) policy or hot cache pollution like the least frequently used (LFU) and SIZE policies [7] because these policies consider just one factor and ignore other factors that influence the efficiency the web caching. Consequently, designing a better-suited caching policy that would improve the performance of the web cache is still an incessant research [6, 8].

Many web cache replacement policies have been proposed attempting to get good performance [2, 9, 10]. However, combination of the factors that can influence the replacement process to get wise replacement decision is not easy task because one factor in a particular situation or environment is more important than others in other environments [2, 9]. In recent years, some researchers have been developed intelligent approaches that are smart and adaptive to web caching environment [2]. These include adoption of back-propagation neural network, fuzzy systems, evolutionary algorithms, etc. in web caching, especially in web cache replacement.

The neuro-fuzzy system is a neural network that is functionally equivalent to a fuzzy inference model. A common approach in neuro-fuzzy development is the adaptive neuro-fuzzy inference system (ANFIS) that has more power than Artificial Neural Networks (ANNs) and fuzzy systems as ANFIS integrates the best features of fuzzy systems and ANNs and eliminates the disadvantages of them.

In this paper, the proposed approach grounds short-term cache that receives the web objects from the Internet directly, while long-term cache receives the web objects from the short-term cache as these web objects visited more than pre-specified threshold value. Moreover, neuro-fuzzy system is employed to predict web objects that can be re-accessed later. Hence, unwanted objects are removed efficiency to make space of the new web objects.

The remaining parts of this paper are organized as follows: literature review is presented in Section 2, related works of intelligent web caching techniques are discussed in Section 2.1. Section 2.2 presents client-side web caching, and Section 2.3 describes neuro-fuzzy system and ANFIS. A framework of Intelligent Client-side Web Caching Scheme is portrayed in Section 3, while Section 4 elucidates the experimental results. Finally, Section 5 concludes the paper and future work.

2. LITERATURE REVIEW

2.1 Related Works on Intelligent Web Caching

Although there are many studies in web caching, but research on Artificial Intelligence (AI) in web caching is still fresh. This section presents some existing web caching techniques based on ANN or fuzzy logic.

In [11], ANN has been used for making cache replacement decision. An object is selected for replacement based on the rating returned by ANN. This method ignored latency time in replacement decision. Moreover, the objects with the same class are removed without any precedence between these objects. An integrated solution of ANN as caching decision policy and LRU technique as replacement policy for script data object has been proposed in [12]. However, the most important factor in web caching, i.e., recency factor, was ignored in caching decision. Both prefetching policy and web cache replacement decision has been used in [13]. The most significant factors (recency and frequency) were ignored in web cache replacement decision. Moreover, applying ANN in all policies may cause extra overhead on server. ANN has also been used in [6] depending on syntactic features from HTML structure of the document and the HTTP responses of the server as inputs. However, this method ignored frequency factor in web cache replacement decision. On other hand, it hinged on some factors that do not affect on performance of the web caching.

Although the previous studies have shown that the ANNs can give good results with web caching, the ANNs have the following disadvantage: ANNs lack explanatory capabilities, the performance of ANNs relies on the optimal selection of the network topology and its parameters, ANNs learning process can be time consuming, and ANNs are also too dependent on the quality and amount of data available [14, 15, 16].

On other hand, [17] proposed a replacement algorithm based on fuzzy logic. This method ignored latency time in replacement decision. Moreover, the expert knowledge may not always available in web caching. This scheme is also not adaptive with web environment that changes rapidly.

This research shares consideration of frequency, recency, size and latency time in replacement decision with some previous replacement algorithms. Neuro-Fuzzy system especially ANFIS is implemented in replacement decision since ANFIS integrates the best features of fuzzy systems and ANNs. On the contrary, our scheme differs significantly in methodology used in caching the web objects, and we concentrate more on client-side caching as it is economical and effective way, primarily due its close proximity to the user [3.4].

2.2 Client-side Web Caching

Caches are found in browsers and in any of the web intermediate between the user agent and the original server. Typically, a cache is located in the browser and the proxy [18]. A browser cache (client-side cache) is located in client. If we examine the preferences dialog of any modern web browser (like Internet Explorer, Safari or Mozilla), we will probably notice a cache setting. Since most users visit the same web site often, it is beneficial for a browser to cache the most recent set of pages downloaded. In the presence of web browser cache, the users can interact not only with the web pages but also with the web browser itself via the use of the special buttons such as back, forward, refresh or via URL rewriting. On other hand, a proxy cache is located in proxy. It works on the same principle, but in a larger scale. The proxies serve hundreds or thousands of users in the same way.

As cache size is limited, a cache replacement policy is needed to handle the cache content. If the cache is full when an object needs to be stored, then the replacement policy will determine which object is to be evicted to allow space for the new object. The goal of the replacement policy is to make the best use of available cache space, to improve hit rates, and to reduce loads on the original server. The simplest and most common cache management approach is LRU algorithm,

which removes the least recently accessed objects until there is sufficient space for the new object. LRU is easy to implement and proficient for uniform size objects such as in the memory cache. However, since it does not consider the size or the download latency of objects, it does not perform well in web caching [6].

Most web browsers still concern traditional replacement policies [3, 4] that are not efficient in web caching [6]. In fact, there are few important factors of web objects that can influence the replacement policy [2, 9, 10]: recency, i.e., time of (since) the last reference to the object, frequency, i.e., number of the previous requests to the object, size, and access latency of the web object. These factors can be incorporated into the replacement decision. Most of the proposals in the literature use one or more of these factors. However, combination of these factors to get wise replacement decision for improving performance of web caching is not easy task because one factor in a particular situation or environment is more important than others in other environments [2, 9].

2.3 Neuro-Fuzzy System and ANFIS

The neuro-fuzzy systems combine the parallel computation and learning abilities of ANNs with the human-like knowledge representation and explanation abilities of fuzzy systems [19]. The neuro-fuzzy system is a neural network that is functionally equivalent to a fuzzy inference model.

A common approach in neuro-fuzzy development is the adaptive neuro-fuzzy inference system (ANFIS), which has shown superb performance at binary classification tasks, being a more profitable alternative in comparison with other modern classification methods [20]. In ANFIS, the membership function parameters are extracted from a data set that describes the system behavior. The ANFIS learns features in the data set and adjusts the system parameters according to a given error criterion. Jang's ANFIS is normally represented by six-layer feed forward neural network [21].

It is not necessary to have any prior knowledge of rule consequent parameters since ANFIS learns these parameters and tunes membership functions accordingly. ANFIS uses a hybrid learning algorithm that combines the least-squares estimator and the gradient descent method. In ANFIS training algorithm, each epoch is composed of forward pass and backward pass. In forward pass, a training set of input patterns is presented to the ANFIS, neuron outputs are calculated on the layer-by-layer basis, and rule consequent parameters are identified. The rule consequent parameters are identified by the least-squares estimator. Subsequent to the establishment of the rule consequent parameters, we compute an actual network output vector and determine the error vector. In backward pass, the back-propagation algorithm is applied. The error signals are propagated back, and the antecedent parameters are updated according to the chain rule. More details are illustrated in [21].

3. FRAMEWORK OF INTELLIGENT WEB CLIENT-SIDE CACHING SCHEME

In this section, we present a framework of Intelligent Client-side Web Caching Scheme. As shown in FIGURE 1, the web cache is divided into short-term cache that receives the web objects from the Internet directly, and long-term cache that receives the web objects from the short-term cache.

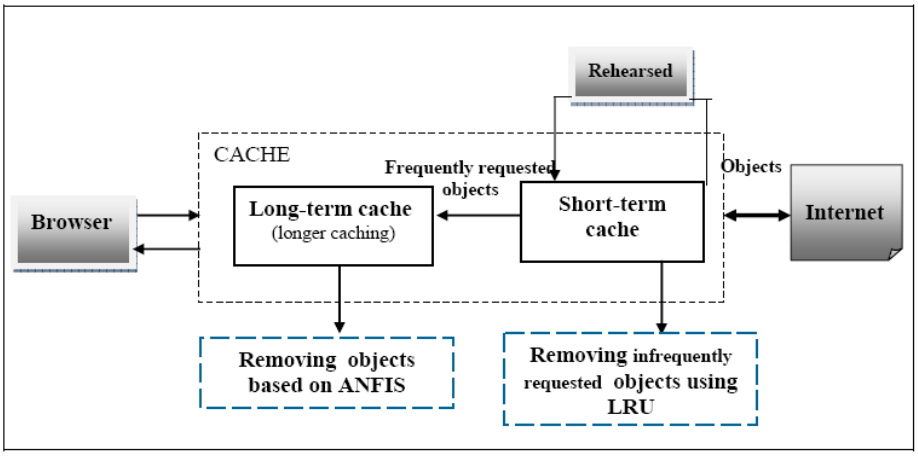

FIGURE 1: A Framework of Intelligent Client-side Web Caching Scheme.

When the user navigates specific web page, all web objects embedded in the page are stored in short-term cache primarily. The web objects that visited more than once will be relocated to long-term cache for longer caching but the other objects will be removed using LRU policy that removes the oldest object firstly. This will ensure that the preferred web objects are cached for longer time, while the bad objects are removed early to alleviate cache pollution and maximize the hit ratio. On the contrary, when the long-term cache saturates, the trained ANFIS is employed in replacement process by classifying each object stored in long-term cache to cacheable or uncacheable object. The old uncacheable objects are removed initially from the long-term cache to make space for the incoming objects (see algorithm in FIGURE 2). If all objects are classified as cacheable objects, then our approach will work like LRU policy.

In training phase of ANFIS, the desired output is assigned to one value and the object considered cacheable object if there is another request for the same object at a later point in specific time only. Otherwise, the desired output is assigned to zero and the object considered uncacheable object.

The main feature of the proposed system is to be able to store ideal objects and remove unwanted objects early, which may alleviate cache pollution. Thus, cache space is used properly. The second feature of the proposed system is to be able to classify objects to either cacheable or uncacheable objects. Hence, the uncacheable objects are removed wisely when web cache is full. The proposed system is also adaptive and adjusts itself to a new environment as it depends on adaptive learning of the neuro-fuzzy system. Lastly, the proposed system is very flexible and can be converted from a client cache to a proxy cache using minimum effort. The difference lies mainly in the data size at the server which is much bigger than the data size at the client.

```
If (size of new object > free space of long-term cache)
{    // update objects' priorities using ANFIS
  For each object(i) in long-term cache
  { The common attributes of object(i) are forwarded as inputs of the trained
      ANFIS.
     If ( out of ANFIS > 0.5)
        priority of object(i)=1;              // cacheable object
     Else
        priority of object(i)=0;              // uncacheable object
      } // end for
  Do
  { // remove the oldest uncachable object
    delete the oldest object that has priority=0 form long-term cache
  } while (size of new object > free space of long-term cache);
If all objects have priority=1
{ Do {
       delete the oldest object form long-term cache
    } while(size of new object > free space of long-term cache);
  }
}
```

FIGURE 2: Intelligent long-term cache removal algorithm based on ANFIS.

4. Experimental Results

In our experiment, we use BU Web trace [22] provided by Cunha of Boston University. BU trace is composed of 9633 files, recording 1,143,839 web requests from different users during six months. Boston traces consist of 37 client machines divided into two sets: undergraduate students set (called 272 set) and graduate students set (called B19 set). The B19 set has 32 machines but the 272 set has 5 machines. In this experiment, twenty client machines are selected randomly from both 272 set and the B19 set for evaluating performance of the proposed method.

Initially, about one month data is used (December for clients from 272 set and January for clients from B19 set) as training dataset for ANFIS. The dataset is divided into training data (70%) and testing data (30%). From our observation; one month period is sufficient to get good training with small Mean Square Error (MSE) and high classification accuracy for both training and testing. The testing data is also used as validation for probing the generalization capability of the ANFIS at each epoch. The validation data set can be useful when over-fitting is occurred.

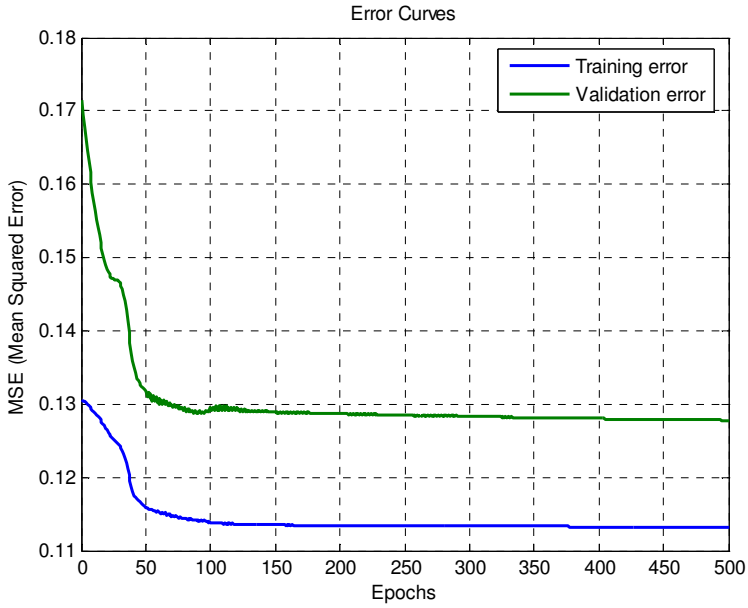

FIGURE 3: Error convergence of ANFIS.

FIGURE 3 shows error convergence of ANFIS for one of client machines, called beaker client machine, in training phase. As shown in FIGURE 3, the network converges very well and produces small MSE with two member functions. It has also good generalization ability as validation error decreases obviously. Thus, it is adequate to use two member functions for each input in training ANFIS. Table 1 summarizes the setting and parameters used for training ANFIS.

ANFIS parameter	Value
Type of input Member function(MF)	Bell function
Number of MFs	2 for each input
Type of output MF	linear
Training method	hybrid
Number of linear parameters	80
Number of nonlinear parameters	24
Total number of parameters	104
Number of fuzzy rules	16
Maximum epochs	500
Error tolerance	0.005

TABLE 1: Parameters and their values used for training of ANFIS.

Initially, the membership function of each input is equally divided into two regions, small and large, with initial membership function parameters. Depending on used dataset, ANFIS is trained to adjust the membership function parameters using hybrid learning algorithm that combines the least-squares method and the back-propagation algorithm. FIGURE 4 shows the changes of the final (after training) membership functions with respect to the initial (before training) membership functions of the inputs for the beaker client machine.

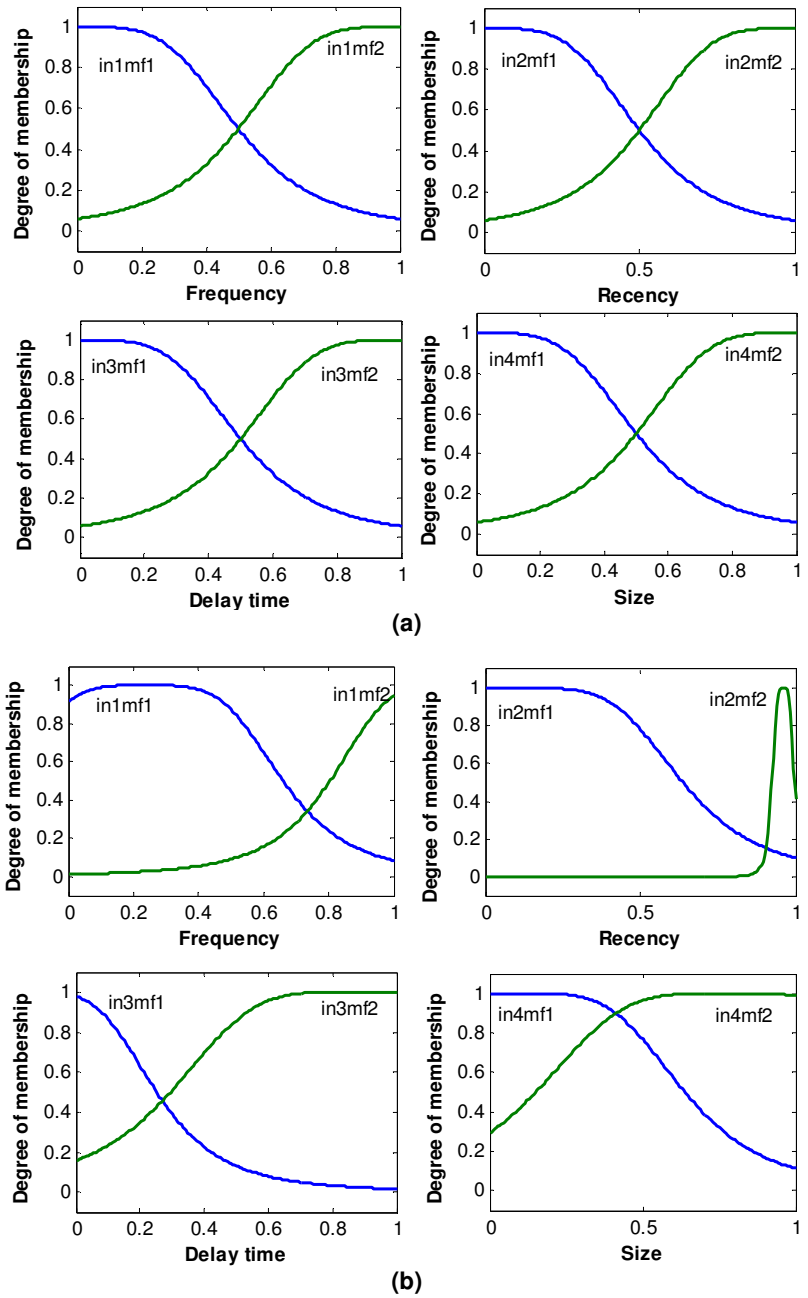

FIGURE 4: (a) Initial and (b) final generalized bell shaped membership functions of inputs of ANFIS.

Since ANFIS is employed in replacement process by classifying each object stored in long-term cache into cacheable or uncacheable object, the correct classification accuracy is the most important measure for evaluating training of ANFIS in this study. FIGURE 5 and FIGURE 6 show the comparison of the correct classification accuracy of ANFIS and ANN for 20 clients in both training and testing data. As can be seen in FIGURE 5 and FIGURE 6, both ANN and ANFIS produce good classification accuracy. However, ANFIS has higher correct classification for both training and testing data in most clients compared to ANN. The results in FIGURE 5 and FIGURE 6 also illustrate that ANFIS has good generalization ability since the correct classification ratios of the training data are similar to the correct classification ratios of the testing data for most clients.

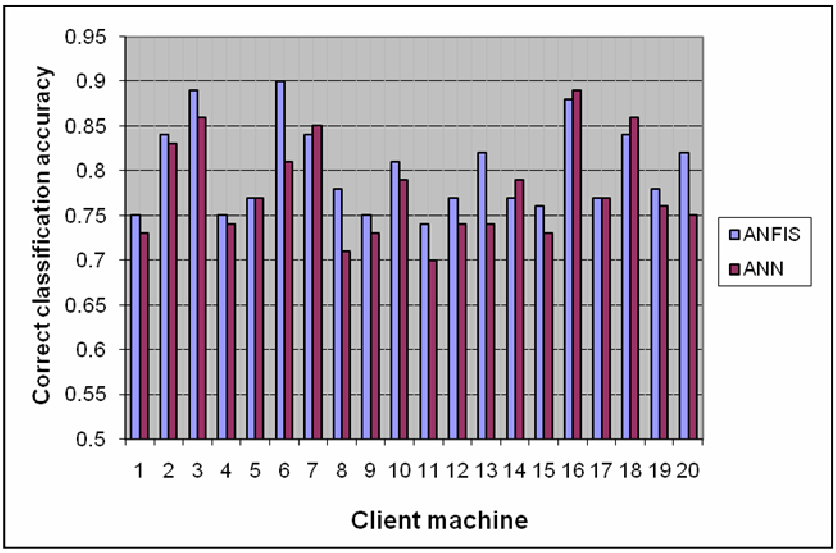

FIGURE 5: Comparison of the correct classification accuracy for training data.

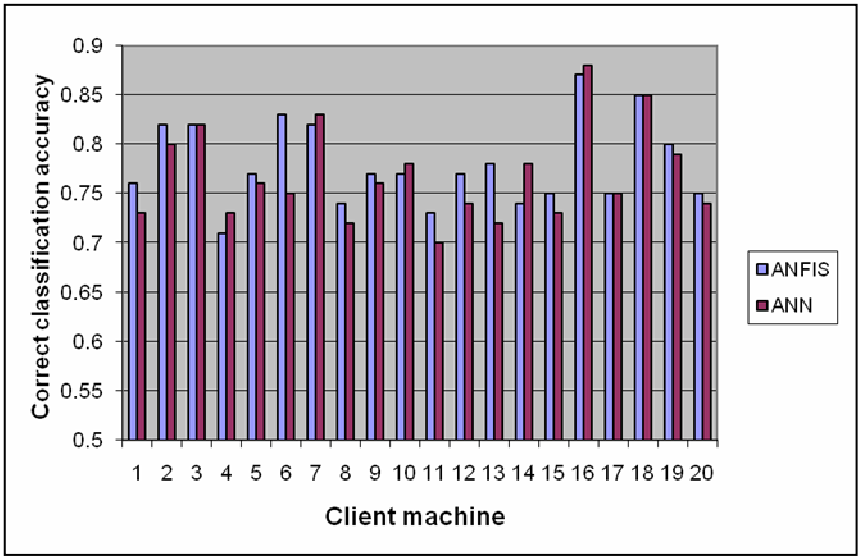

FIGURE 6: Comparison of the correct classification accuracy for testing data.

To evaluate the performance of the proposed method, trace-driven simulator is developed (in java) which models the behavior of Intelligent Client-side Web Caching Scheme. The twenty clients' logs and traces for next two months of the training month are used as data in trace-driven simulator. BU Traces do not contain any information on determining when the objects are unchanged. To simulate correctly an HTTP/1.1 cache, size is a candidate for the purpose of consistency. Thus, the object that has the same URL but different size is considered as the updated version of such object [23].

Hit ratio (HR), Byte hit ratio (BHR) and Latency saving ratio (LSR) are the most widely used metrics in evaluating the performance of web caching [6,9]. HR is defined as the percentage of requests that can be satisfied by the cache. BHR is the number of bytes satisfied from the cache as a fraction of the total bytes requested by user. LSR is defined as the ratio of the sum of download time of objects satisfied by the cache over the sum of all downloading time.

In the proposed method, an obvious question would be the size of each cache. Many experiments were done to show the best size of each cache to ensure better performance. The simulation results of hit ratio of five clients with various sizes of short-term cache are illustrated in FIGURE 7. The short-term cache with size 40% and 50% of total cache size performed the best performance. Here, we assumed that the size of the short-term cache is 50% of the total cache size.

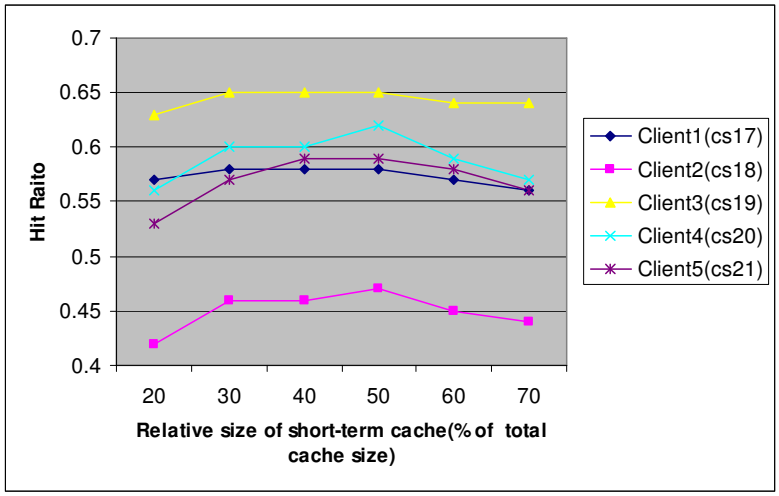

FIGURE 7: Hit Ratio for different short-term cache sizes.

For each client, the maximum HR, BHR, and LSR are calculated for a cache of infinite size. Then, the measures are calculated for a cache of size 0.1%, 0.5%, 1%, 1.5% and 2% of the infinite cache size, i.e., the total size of all distinct objects requested, to determine the impact of cache size on the performance measures accordingly. We observe that the values are stable and close to maximum values after 2% of the infinite cache size in all policies. Hence, this point is claimed as the last point in cache size.

The performance of the proposed approach is compared to LRU and LFU policies that are the most common policies and form the basis of other web cache replacement algorithms [11]. FIGURE 8, FIGURE 9 and FIGURE 10 show the comparison of the average values of HR, BHR and LSR for twenty clients for the different policies with varying relative cache size. The HR, BHR and LSR of the proposed method include HR, BHR and LSR in both short-term cache and the long-term cache.

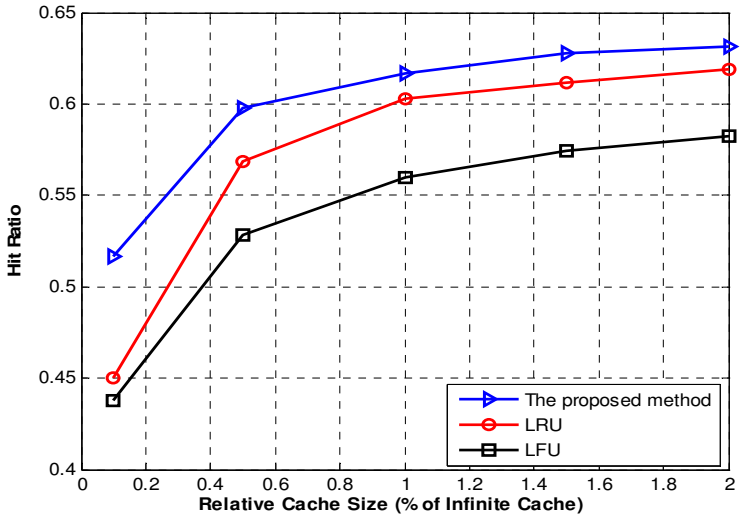

FIGURE 8: Impact of cache size on hit ratio.

As can be seen in FIGURE 8, when the relative cache size increases, the average HR boosts as well for all algorithms. However, the percentage of increasing is reduced when the relative cache size increases. When the relative cache size is small, the replacement of objects is required frequently. Hence, the impact of the performance of replacement policy appears clearly. The results in FIGURE 8 clearly indicate that the proposed method achieves the best HR among all algorithms across traces of clients and cache sizes. This is mainly due to the capability of the proposed method in storing the ideal objects and removing the bad objects that predicted by ANFIS.

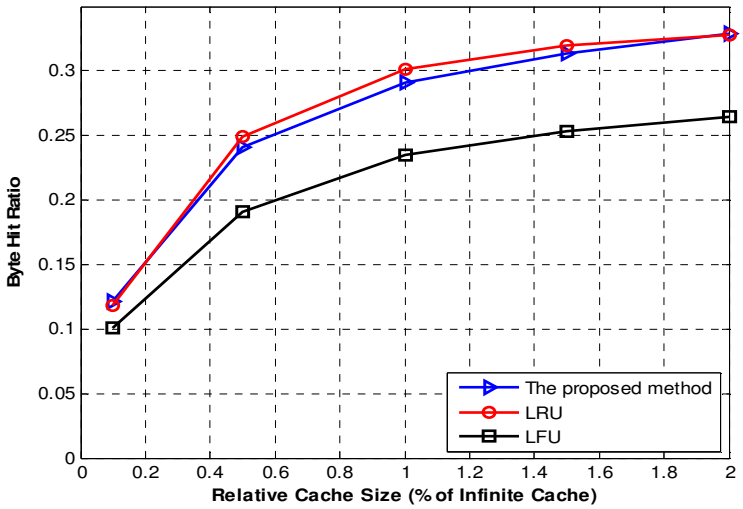

FIGURE 9: Impact of cache size on byte hit ratio.

Although the proposed approach has superior performance in terms of HR compared to all other policies, it is not surprising that BHR of the proposed method is similar or slightly worse than LRU (FIGURE 9). This is because of training of ANFIS with desired output depends on the future request only and not in terms of a size-related cost. These results conform to findings obtained by [11] since we use the same concept as [11] to acquire the desired output in the training phase.

The results also indicate that our approach concerns with ideal small objects to be stored in the cache that usually produces higher HR.

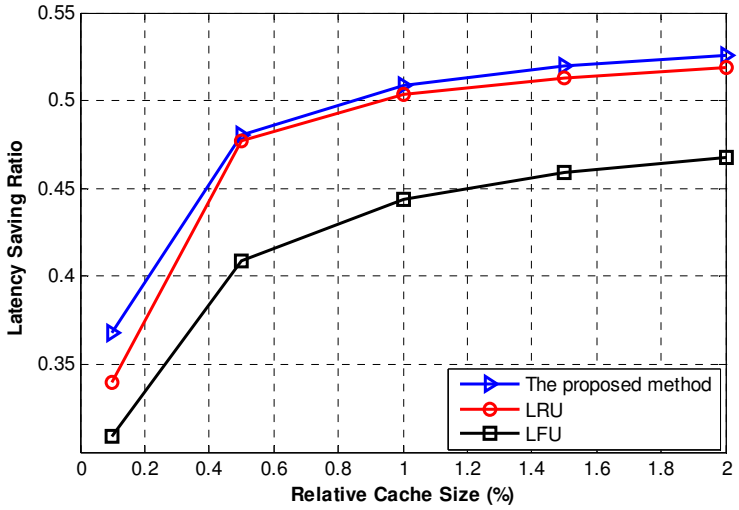

FIGURE 10: Impact of cache size on latency saving ratio.

FIGURE 10 illustrates the average LSR of the proposed method and the common caching schemes as the function of cache sizes. FIGURE 10 depicts that LSR increases rapidly for all policies. However, the proposed method outperforms others policies. LSR of the proposed approach is much better than LFU policy in all conditions. Moreover, LSR of the proposed approach is significantly better than LSR of LRU when the cache sizes are small (less or equal than 0.5% of the total size of all distinct objects). In these situations, many replacements occur and good replacement algorithm is important. On other hand, LSR of the proposed is slightly higher than LSR of LRU when the cache size is larger. Although the desired output through the training phase concern just on future request regardless delay-related cost, the proposed method outperforms the other policies in terms of LSR. This is a result of the close relationship between HR and LSR. Many studies have shown that increasing of HR improves LSR [24, 25].

In all conditions, LFU policy was the worst in all measures because of the cache pollution in objects with the large reference accounts, which are never replaced even if they are not re-accessed again. From FIGURE 8, FIGURE 9 and FIGURE 10, percents of improvements of the performance in terms of HR, BHR and LSR achieved by our approach over the common policies can be concluded and summarized as shown in TABLE 2.

Relative Cache Size (%)	Percent of Improvements (%)					
	LRU			LFU		
	HR	BHR	LSR	HR	BHR	LSR
0.1	14.8	2.57	8.3	17.9	20.30	18.9
0.5	5.2	-	0.81	13.3	26.25	17.48
1	2.32	-	1.04	10.2	24.04	14.53
1.5	2.58	-	1.41	9.32	24.11	13.29
2	1.96	0.38	1.35	8.33	20.30	12.5

TABLE 2: Performance improvements achieved by the proposed approach over common policies.

The results in TABLE 2 indicate that the proposed approach can improve the performance in terms of HR up to 14.8% and 17.9%, in terms of BHR up to 2.57% and 26.25%, and in terms of

LSR up to 8.3% and 18.9% compared to LRU and LFU respectively. TABLE 2 also shows that no improvements of BHR achieved by the proposed approach over LRU policy with relative cache sizes: 0.5%, 1%, and 1.5%.

5. CONCLUSION & FUTURE WORK

Web caching is one of the effective solutions to avoid Web service bottleneck, reduce traffic over the Internet and improve scalability of the Web system. This study proposes intelligent scheme based on neuro-fuzzy system by splitting cache to two caches, short-term cache and long-term cache, on a client computer for storing the ideal web objects and removing the unwanted objects in the cache for more effective usage. The objects stored in short-term cache are removed by LRU policy. On other hand, ANFIS is employed to determine which web objects at long-term cache should be removed. The experimental results show that our approach has better performance compared to the most common policies and has improved the performance of client-side caching substantially.

One of the limitations of the proposed Intelligent Client-side Web Caching Scheme is complexity of its implementation compared to LRU that is very simple. In addition, the training process requires extra computational overhead although it happens infrequently. In the real implementation, the training process should be not happened during browser session. Hence, the user does not fell bad about this training. In recent years, new solutions have been proposed to utilize cache cooperation on client computers to improve client-side caching efficiency. If a user request misses in its local browser cache, the browser will attempt to find it in another client's browser cache in the same network before sending the request to proxy or the original web server. Therefore, our approach can be more efficient and scalable if it supports mutual sharing of the ideal web object stored in long-term cache.

Acknowledgments. This work is supported by Ministry of Science, Technology and Innovation (MOSTI) under eScience Research Grant Scheme (VOT 79311). Authors would like to thank Research Management Centre (RMC), Universiti Teknologi Malaysia, for the research activities and *Soft Computing Research Group* (SCRG) for the support and incisive comments in making this study a success.

6. REFERENCES

1. L. D. Wessels. *"Web Caching"*. USA: O'Reilly. 2001.

2. H.T. Chen. *"Pre-fetching and Re-fetching in Web Caching systems: Algorithms and Simulation"*. Master Thesis, TRENT UNIVESITY, Peterborough, Ontario, Canada, 2008.

3. V. S. Mookerjee, and Y. Tan. *"Analysis of a least recently used cache management policy for Web browsers"*. Operations Research, Linthicum, Mar/Apr 2002, Vol. 50, Iss. 2, p. 345-357.

4. Y. Tan, Y. Ji, and V.S Mookerjee. "*Analyzing Document-Duplication Effects on Policies for Browser and Proxy Caching"*. INFORMS Journal on Computing. 18(4), 506-522. 2006.

5. T. Chen. *" Obtaining the optimal cache document replacement policy for the caching system of an EC website"*. European Journal of Operational Research, Amsterdam, Sep 1, 2007, Vol. 181, Iss. 2; p. 828.

6. T.Koskela, J.Heikkonen, and K.Kaski. *"Web cache optimization with nonlinear model using object feature"*. Computer Networks journal, elsevier , 20 December 2003, Volume 43, Number 6.

7. R .Ayani, Y.M. Teo, and Y. S. Ng. *"Cache pollution in Web proxy servers".* International Parallel and Distributed Processing Symposium (IPDPS'03). ipdps, p. 248a,2003.

8. C. Kumar, and J.B Norris. "*A new approach for a proxy-level Web caching mechanism. Decision Support Systems"*. 46(1), 52-60. Elsevier Science Publishers. 2008.

9. A.K.Y. Wong. *"Web Cache Replacement Policies: A Pragmatic Approach".* IEEE Network magazine, 2006 , vol. 20, no. 1, pp. 28–34.

10. S.Podlipnig, and L.Böszörmenyi. *"A survey of Web cache replacement strategies"*. ACM Computing Surveys, 2003, vol. 35, no. 4, pp. 374-39.

11. J.Cobb, and H.ElAarag. *"Web proxy cache replacement scheme based on back-propagation neural network"*. Journal of System and Software (2007),doi:10.1016/j.jss.2007.10.024.

12. Farhan. *"Intelligent Web Caching Architecture".* Master thesis, Faculty of Computer Science and Information System, UTM University, Johor, Malaysia, 2007.

13. U.Acharjee. *"Personalized and Artificial Intelligence Web Caching and Prefetching"*. Master thesis, Canada: University of Ottawa, 2006.

14. X.-X . Li, H .Huang, and C.-H .Liu.*" The Application of an ANFIS and BP Neural Network Method in Vehicle Shift Decision"*. 12th IFToMM World Congress, Besançon (France), June18-21, 2007.M.C.

15. S.Purushothaman, and P.Thrimurthy. "*Implementation of Back-Propagation Algorithm For Renal Data mining".* International Journal of Computer Science and Security. 2(2), 35-47.2008.

16. P. Raviram, and R.S.D. Wahidabanu. "*Implementation of artificial neural network in concurrency control of computer integrated manufacturing (CIM) database"*. International Journal of Computer Science and Security. 2(5), 23-35.2008.

17. Calzarossa, and G.Valli. *"A Fuzzy Algorithm for Web Caching".* SIMULATION SERIES journal, 35(4), 630-636, 2003.

18. B. Krishnamurthy, and J. Rexforrd. *"Web protocols and practice: HTTP/1.1, networking protocols, caching and traffic measurement".* Addison-Wesley, 2001.

19. Masrah Azrifah Azmi Murad, and Trevor Martin. "*Similarity-Based Estimation for Document Summarization using Fuzzy Sets"*. International Journal of Computer Science and Security. 1(4), 1-12. 2007.

20. J. E. Muñoz-Expósito,S. García-Galán,N. Ruiz-Reyes, and P. Vera-Candeas. *"Adaptive network-based fuzzy inference system vs. other classification algorithms for warped LPC-based speech/music discrimination"*. Engineering Applications of Artificial Intelligence,Volume 20 , Issue 6 (September 2007), Pages 783-793,Pergamon Press, Inc. Tarrytown, NY, USA, 2007.

21. Jang. *"ANFIS: Adaptive-network-based fuzzy inference system"*. IEEE Trans Syst Man Cybern 23 (1993) (3), pp. 665.

22. BU Web Trace,http://ita.ee.lbl.gov/html/contrib/BU-Web-Client.html.

23. W. Tian, B. Choi, and V.V. Phoha . "*An Adaptive Web Cache Access Predictor Using Neural Network"*. Published by :Springer- Verlag London, UK. 2002.

24. Y. Zhu, and Y. Hu. "*Exploiting client caches to build large Web caches"*. The Journal of Supercomputing. 39(2), 149-175. Springer Netherlands. .2007.

25. L. SHI, L. WEI, H.Q. YE, and Y.SHI. "*MEASUREMENTS OF WEB CACHING AND APPLICATIONS"*. Proceedings of the Fifth International Conference on Machine Learning and Cybernetics, 13-16 August 2006. Dalian,1587-1591.