A Two-Level Intelligent Web Caching Scheme with a Hybrid Extreme Learning Machine and Least Frequently Used

Phet Imtongkhum[1], Chakchai So-In[1], Surasak Sanguanpong[2], Songyut Phoemphon[1]

[1] Department of Computer Science, Faculty of Science, Khon Kaen University, Thailand
[2] Department of Computer Engineering, Faculty of Engineering, Kasetsart University, Thailand
phet@kkumail.com, chakso@kku.ac.th, surasak.s@ku.ac.th, songyut_p@kkumail.com

Abstract

The immense increase in data traffic has created several issues for the Internet community, including long delays and low throughput. Most Internet user activity occurs via web access, thus making it a major source of Internet traffic. Due to a lack of effective management schemes, Internet usage is inefficient. Advances in caching mechanisms have led to the introduction of web proxies that have improved real-time communication and cost savings. Although several traditional caching polices have been implemented to increase speed and simplicity, cache replacement accuracy remains a key limitation due to cache storage constraints. Our contribution concerns the algorithmic investigation of intelligent soft computing schemes to enhance a web proxy system to improve precision for reproducibility. This research also proposes a two-level caching scheme; the first level is least frequently used (LFU), and an extreme learning machine (ELM) is used for the second level. A traditional ELM for web caching is further optimized with object similarity factors. The proposed scheme is evaluated and compared to a traditional caching policy and its integration with intelligent caching using a well-known dataset from IRCache. The method is shown to achieve good performance in terms of high hit and byte hit rates.

Keywords: Extreme learning machine, Least frequently used, Proxy, Replacement, Web caching

1 Introduction

Internet technology is now accessible around the world, leading to widespread Internet use and increased Internet traffic. According to a 2015 report by the International Telecommunication Union (ITU), the Internet community includes more than 3.2 billion people, and usage continues to increase [1]. This rapid increase has led to traffic of up to 72.5 exabytes per month [2], of which more than 98% is web-based Internet traffic [3]. The infrastructure of the Internet has not been able to keep pace with growth because of limited investment by Internet Service Providers (ISPs) [4-5].

Consequently, many studies and applications have been developed to not only evaluate policy-based approaches of multi-tier ISPs but also efficiently use the existing Internet infrastructure, particularly from the user perspective, including factors such as the throughput and delay, which contribute to the quality of accessible web objects. Web proxies (web caching) have been developed to address this issue. The key concept that underlies proxy/caching is similar to that of traditional caching of the memory hierarchical structure of a CPU, including the CPU cache, RAM, and hard disk [6]. The inter-memory levels are positioned such that the time required to acquire data due to speed and distance is reduced. The efficacy of this structure can also be increased by accessible data repetition. This perspective is then applied to caching web objects to reduce the access time for long-distance Internet use, including bandwidth optimization with repetitive accessible patterns of Internet users.

There are two types of proxies: forward and reverse [5]. The key difference between these proxies is based on their objectives and locations. A forward proxy is placed between the clients and the Internet (outbound) to reduce the delay in acquiring web objects that are far from the client location. The sub-sequence client, if present, can directly retrieve the objects from the proxy instead of re-traversing to the original source of the web objects. In contrast, a reverse proxy is positioned in front of the server farms that are used for balancing heavy loads from large accesses, including computational tasks to reproduce a particular web object within the proxy, and directly forwards the objects to the requester without the need for re-computation at the server.

Web caching has been applied by numerous organizations and providers; however, there are several trade-offs such as the need for additional hardware and software for cache management and extra computational processes. In addition, caching can only be applied to static data access. Due to these constraints, particularly caching storage, web caching further requires a smart

*Corresponding Author: Chakchai So-In; E-mail: chakso@kku.ac.th
DOI: 10.3966/160792642018051903010

caching replacement policy [7] to achieve higher utilization of bandwidth as well as short delays. For example, a superior replacement algorithm should select non-repetitive objects from the cache to minimize space (caching storage).

Generally, a traditional web caching replacement policy (TCRP) simply applies a single (or a few) factor(s), such as timing, ordering, frequency, or size, to determine the object replacement. The key advantage of a TCRP is its speed due to its low complexity, which makes TCRPs suitable for real-time or online communication [8]. The trade-off, however, of a TCRP is the replacement precision, which will probably lead to the un-optimized use of caching storage and thus high cost and larger budgets.

Soft computing has recently been proposed to resolve the problem of uncertainty and a non-linear solver. Soft computing is used in science and engineering problems [9], such as clustering and classification, and particularly in the areas of recognition and prediction, including web caching replacements [10]. Several classes of soft computing, such as Neural Networks (NNs), Fuzzy Logic (FL), Support Vector Machines (SVMs), and Evolutionary Computation (EC), can be applied to a particular problem based on the characteristics of the soft computing; their key advantages are their high precision with comparable computational complexity trade-offs with heuristic approaches.

Consequently, in addition to providing a detailed discussion of how to apply other applications of soft computing to web caching algorithms for reproducibility, this research proposes a combination of the two approaches (i.e., high speed and high precision) in the form of a two-level web caching scheme. To achieve the high-speed retrieval of web objects for clients, a TCRP is selected for the first (small) level of caching storage.

To achieve higher precision with a timing trade-off, soft computing approaches are then applied as the second level for (larger) caching storage. The TCRP is investigated to select the best algorithm, although it is not limited to this algorithm; we then select an extreme learning machine (ELM) from several traditional soft computing techniques for use as the second level replacement policy. An enhancement of the ELM with respect to similarity is also proposed; these methods are denoted Hybrid ELM-LFU (H-ELM-LFU).

This article is organized as follows. Section 2 provides a brief overview of web caching schemes and particularly traditional caching replacement and intelligent caching classification. Section 3 explains the techniques of the present work, including the use of intelligent caching and its integration. The concept of our proposed two-level caching scheme, including an ELM and its enhancement, is discussed in Section 4. The performance of our proposed scheme is compared with other state-of-the-art soft computing applications with traditional caching replacement policies in Section 5. Finally, Section 6 offers conclusions and possible future research.

2 Overview of Web Caching Replacement Schemes

This section provides a brief overview of web caching replacement schemes categorized into two classes: traditional caching replacement policies and intelligent caching schemes. However, the focus is on soft-computing-based schemes.

2.1 Traditional Caching Replacement Policy (TCRP)

Although several caching schemes have been proposed, e.g., Bélády, Most Recently Used, Random Replacement, Low Inter-reference Recency Set, and Adaptive Replacement Cache [11], the focus here is on well-known (primitive) caching techniques applied to the context of web caching, including First In First Out (FIFO), Least Recently Used (LRU), Least Frequently Used (LFU), and Greedy Dual Size (GDS). Note that several of their derivatives, such as LRU-MIN and Hierarchical GD, are also available, which also results in greater computational work; however, again, each of these methods can be applied for the selection of the first-level cache for our proposed hybrid model.

These schemes have been adopted from traditional time scheduling, which is used in CPUs, the queuing problem, and particularly in high-speed queuing applications [6]. These schemes apply simple statistical analyses, and their key advantage is their simplicity (i.e., low computational complexity) such as from considering only a single (or only a few) factor(s)/ parameter(s).

FIFO can be considered a pioneering scheduling or replacement scheme that is based on arrival time. In other words, the oldest web object (earliest timestamp) will be selected for replacement first, and this process is iterative based on the time domain.

LRU considers the time to access a particular web object in the final replacement decision instead of replacement ordering. This algorithm assumes that recent objects will be frequently used.

LFU assumes that the greater the number of accessible objects, the higher the probability of being cached. LFU applies the accumulative frequency of web access to make the replacement decision; low-frequency objects will be selected first for replacement.

GDS can be considered a functional-based model that utilizes two factors: cost and size. The relationship between these two factors is given by the equation below, where H denotes the priority factor of replacement, c (cost) is a co-efficiency factor that ranges from 0 to 1 and s is the size of a particular web object (e.g., bytes), The lower the value of H, the

higher the probability of replacement.

$$H = \frac{c}{s} \quad \textbf{(1)}$$

2.2 Soft Computing for Web Caching Replacement

As previously discussed, soft computing has recently been optimized to solve science and engineering problems [9] and is particularly suitable as a non-linear solver for uncertainty. There are several classes of soft computing (e.g., NN, FL, SVM, and EC [12-15]), each of which has several derivatives; for example, Genetic Algorithms (GAs), Differential Evolution (DE), and Ant Colony Optimization (ACO) are representatives of the EC class. Each solution is applicable to many problems.

Note that the focus of this study is on the practical integration of soft computing with web cache replacement. Consequently, the discussion is based on a well-known approach (available) that uses a representation of each class: NN for Multilayer Perceptron (MLP) with Back Propagation (BPNN), FL, GA, and SVM.

Neural network (NN) for web caching replacement. An NN is a mathematical model that is used to imitate the human brain in performing an intelligent task [12]; it integrates computational units (neurons) in multiple layers, where the layers are connected by adjustable weights. Traditionally, there are three layers: the input, hidden, and output layers. This research applied MLP with a BPNN (Figure 1).

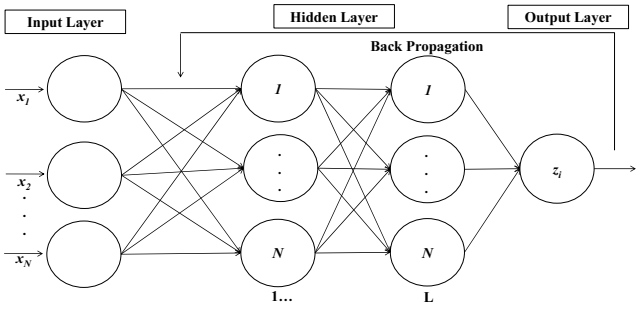


Figure 1. A neural network for web caching

An NN includes two phases: training and testing (this also applies to SVMs and our proposed ELMs, including their derivatives, as will be discussed below). The training phase is conducted to retrieve the weights of the input (*w_in*) and output (*w_out*) with the constraint of the least mean square error (LMSE) given as the objective function *E(t),* as stated in the following equation:

$$E(t) = \sum_{k=1}^{K} \sum_{i=1}^{N(L)} (e_i(k)^2) = \sum_{k=1}^{K} \sum_{i=1}^{N(L)} (d_i(k) - z_i(k))^2 \quad \textbf{(2)}$$

where K denotes the number of epochs (rounds), $N(L)$ is the number of hidden nodes in the L^{th} layer, $d_i(k)$ denotes the target value (actual value) of node i, and z is an output value, such as $y \times w\_out$, in which y is a function of the input x, *w_in*, and *bias*. For the testing, given the stored weights from the training process, i.e., *w_in* and *w_out*, the cacheable probability of the cache being replaced (*CA*) is computed using the equation below, where the input *X_test*, i.e., the HTTP request method (*R*), UID (*U*), and size (*S*), are the testing input values.

$$w\_out \times \left(\frac{1}{1+\exp(-(X\_test \times w\_in))}\right)^T \quad \textbf{(3)}$$

Algorithm 1 and Algorithm 2 show the web cache replacement scheme when the NN is applied in both the training and testing phases. For the training, *w_in* and *w_out* are first randomly generated (line 1) in the range of *-1* to *1* and are then updated based on the LMSE. The inputs *X(R, U, S)* of the web object attributes are transformed to the values in the range of [-1, 1] for *R* (such that *GET* is *1* and the others are *-1*) and the range of [0, 1] for *U* and *S* divided by 10^N where *N* is the number of digits. These forms are then computed based on an activation function (a sigmoid function is used here) to generate the output by multiplying by the input weight *w_in* (line 4). The cacheable probability (*CA*) will be estimated with the output weight *w_out* (line 5).

Algorithm 1: NN in the Training Phase

Input: $X(R,U,S)_N, Y_N, K, N$

Output: $w_{inN,N(L)}, w_{outN(L)}$

1. Generate Random Weight $(w_{inN,N(L)}, w\_out_{N(L)})$ in the range [-1, 1]
2. **WHILE** (*K epoches*)
3. WHILE (*N records*)
4. Calculate Output from Activation Function
$y[j]_{N(L)} \rightarrow sigmoid\{X(R,U,S)[j]_N \times w\_in_{N+1,N(L)}\}$
5. Calculate Cacheable Probability
$CA = y[j]_{N(L)} \times w\_out_{N(L)}$
6. Calculate Cacheable Error
$error[j] = CA - Y[j]$
7. Adjust Output Weight
$w\_out_{N[L]} - (error[j] \times y_{N[L]})^T$
8. Adjust Input Weight
$w\_in_{N,N(L)} - \{error[j] \times w\_out_{N(L)}{}^T\} \times (1-(y[j]^2)) \times X(R,U,S)_N\}^T$
9. **ENDWHILE**
10. **ENDWHILE**

Algorithm 2: NN in the Testing Phase

Input: $X_{test(R,U,S)_N}, w_{in_{N,N(L)}}, w_{out_{N(L)}}$

Output: *CA*

1. Calculate Activation Function

$$y_{N(L)} = sigmoid(X_{test(R,U,S)_N} \times w_{in_{N,N(L)}})$$

2. Calculate Estimated Cacheable Probability

$$CA = w\_out_{N(L)} \times (y_{N(L)})^T$$

Next, the approximated error is computed from the CA and the actual target (*Y*) (line 6). The output weight will then be adjusted based on the error (line 7) as well as the input weight (line 8). These processes will iterate for *K* epochs, leading to the final appropriate weights.

Several steps can be used to test the NN (with a fast testing process) based on the trained inputs (Algorithm 1) as shown in Algorithm 2. First, the testing web object attributes are fed into the activation function and multiplied by the input weight *w_in* (lines 1-2), the approximation of the *CA* is derived from the output weight, and the result is derived from the activation function. The *CA* will be used again as a decision for caching; a value greater than *0.5* indicates caching and vice versa.

Fuzzy logic system for web caching replacement. A Fuzzy Logic System (FLS) is used to manage reasoning in which there is an approximation that does not provide the exact solution. In general, the true value will be in the range from *0* to *1*. The four main processes are described below. The example corresponds to the Mamdani fuzzy system due to its key advantage of simplicity as our selection criteria [16].

A fuzzifier is used to transform the input data mapping to the defined membership functions to determine the crossing points of each function such as Gaussian, trapezoidal, triangular, generalized bell, and sigmoid functions.

A fuzzy rule is used to create the mapping between the input and output given its membership function. For example, for the triangular function and twelve pre-defined rules (Table 1), Figure 2 to Figure 5 show three inputs (Frequency - *F*, Time - *T*, and Size - *S*) and the input (weight) of fuzzy systems using the rule-based approach shown in Table 1: very high (VHI), high (HIG), medium (MED), low (LOW), and very low (VLO) [17].

Table 1. Examples of fuzzy rules

IF Frequency	AND Time	AND Size	THEN Weight
LOW	VHI	MED	VHI
LOW	HIG	HIG	VHI
MED	VHI	HIG	VHI
LOW	VHI	HIG	VHI
LOW	HIG	HIG	HIG
MED	HIG	LOW	MED
MED	VHI	MED	HIG
MED	HIG	HIG	HIG
HIG	VLO	HIG	LOW
HIG	HIG	HIG	LOW
LOW	MED	HIG	HIG
MED	HIG	MED	MED

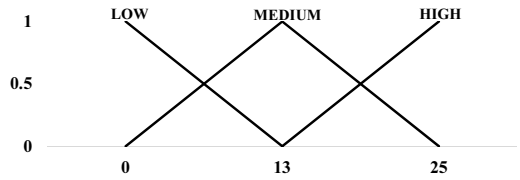


Figure 2. A fuzzy membership function (Frequency)

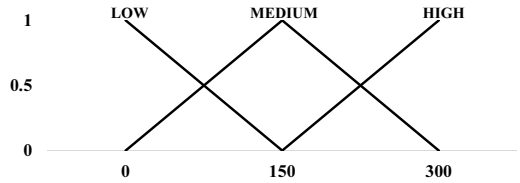


Figure 3. A fuzzy membership function (Time)

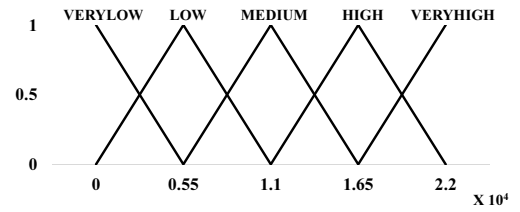


Figure 4. A fuzzy membership function (Size)

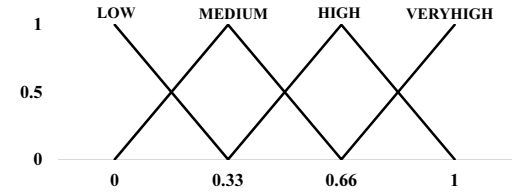


Figure 5. A triangular function (Weight)

A fuzzy inference engine is used to derive the output (weight) given the inputs, such as the size, time, frequency and rule-based weight with aggregation methods such as intersection operations. The output (weight) here is the Replacement Probability (*RP*).

A defuzzifier is used to compute the output based on the Center of Gravity (CoG) over the derived output weight (*RP*). This output (in the range between *0* and *1*)

is then used as the *CA* because if the output is greater than *0.5* [17], the representation of the cache status is "cacheable"; otherwise, it is "uncacheable" or to be replaced.

Algorithm 3 shows an example methodology for web caching using a fuzzy system. First, a set of interesting parameters are considered as a threshold; we consider the entire sets but only three inputs: size, time, and frequency. The membership function that corresponds to these inputs and the input weight (*w_in*) are then generated using the fuzzy rule (lines 1-3). The fuzzy inference engine is then applied with the aggregation method (line 4). Once the output is generated (*RP*), the CoG is computed, which results in the output (*out*) (line 5). Finally, this output is used to indicate the caching stage (*CA*) (line 6).

Algorithm 3: Fuzzy Logic for Web Cache Replacement

Input: *X(S, T, F)*

Output: *CA*

1. Generate Membership Function for the Inputs *X(S, T, F)*
2. Generate Membership Function for input weight *w_in*
3. Generate Fuzzy Rule *(S, T, F, w_in)*
4. Apply Fuzzy Inference Engine (FIE) with Aggregation $RP \leftarrow FIE(S, T, F, w\_in)$
5. Calculate Output (*out*) from the CoG
6. Calculate *CA* based on *out*

Genetic algorithms for web caching replacement. GAs are traditional ECs [18] and include three main steps as follows:

Population selection is used to properly select the population members with a specific size as the parent of the current generation; here, the selection for replacement occurs from the top ten least accumulative frequencies (provided by the ranking algorithm).

The genetic operator is then used to generate better children. There are two common operators. The first one is "crossover", an operator that functions as the blender of two different genes with a common cut (cross) to create different genes. Here, the encoding is the representation of the URL (including the full path) with the cut on only the domain path (domain and domain suffix, including the subdomain if available), as shown in Figure 6.

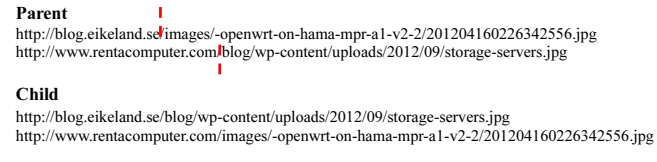


Figure 6. Genetic crossover operator for web caching

The second operator is "mutation", which is used to randomly generate a new population. Here, our encoding is used to randomly replace the first tier domain with other domains, from .com to the top ten domains (*gTLD*), including .net, .org, .info, .biz, .us, .xyz, .mobi, .asia, and .club (see Figure 7) [19].

Figure 7. Genetic mutation operator for web caching

Replacement is used to update the population - the children after the second stage replace those in the scaled population given the utility (i.e., lowest frequency).

Algorithm 4 illustrates the methodology for web cache replacement using the GA with the output of the URL (to be replaced if it exists). First, the URL population is selected based on LFU (i.e., the top ten least access-providing ranking algorithms (lines 1). Next, the crossover operator is applied to create two possible children from the full URL with the concept of the only domain section (URL_{Domain}) and its path (URL_{Domain_Path}) (line 2). The mutation operator is subsequently applied by creating the URL with the first tier domain name from other names from the top ten *gTLD* (line 3). The generated URL is compared with the URL in the caching storage if it exists as the first candidate for replacement.

Algorithm 4: Genetic Algorithm for Web Cache Replacement

Input: URL_s

Output: *URL*

1. Initial Population (URL) with a specific threshold
   new population cache ≤ Top 10 LFU (descending orders)
2. Perform Crossover Operator
   $URL[1]\ CROSS\ URL[2] =$
   $(URL_{Domain}[1], URL_{Domain\_Path}[2])$ and
   $(URL_{Domain}[2], URL_{Domain\_Path}[1])$
3. Perform Mutation Operator based on *x* and *y* within Top 10 gTLD
   $URL_{Domain}[x], URL_{Domain\_Path}[x] \rightarrow URL_{Domain}[y], URL_{Domain\_Path}[x]$

Support vector machine for web caching replacement. An SVM is a supervised learning method that is used for classification. A set of training data will be mapped into two classes by an SVM, which makes it a non-probabilistic binary linear classifier (*-1* or *+1*) [20]. Here, each class represents the stage of caching: cache or not cache (to be replaced). An SVM is generally

modeled as a representation of points in space mapped to separate the category that has the highest margin.

Figure 8 and Figure 9 show a representation of a linear classification given the transformation from an *N*-dimensional input vector *X* to feature vectors that provide the objective function *D(X)* using equation (4) below.

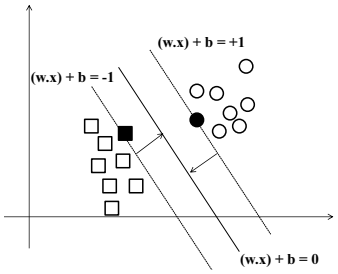


Figure 8. SVM linear classification

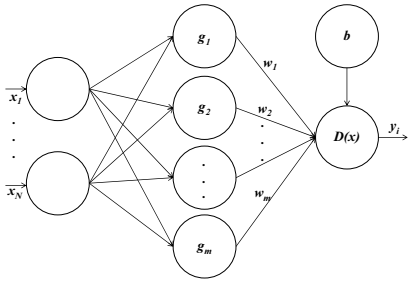


Figure 9. SVM structure

$$D(X) = W \cdot X + W0 = sign\left\{\sum_{j=1}^{m} w_j g(x) + b\right\} \qquad \textbf{(4)}$$

Where the input $X = \{x_1 \ldots x_N\}$ will be fed into the model (HTTP request method - *R*, UID - *U*, and size - *S*) with weights $W = \{w_1 \ldots w_m\}$ such that *N* is the total number of input nodes, *m* is the number of kernel function nodes, and *b* denotes the bias. To derive the weight, Lagrange multipliers will be applied in terms of α (a scaling factor that is the gradient of the function for finding the largest or smallest value). Here, y_i is the target value in the range of [-1, +1] (either hit or miss). *g(x)* denotes an SVM kernel function such as a linear function, polynomial function, radial basis function (RBF), or sigmoid function.

Algorithm 5 shows the SVM (classification) in the training phase. Here, we apply *R*, *U*, and *S* as inputs into the kernel function (line 1), which is the RBF in this case. Here, gamma (γ) is $\frac{1}{2\sigma^2}$ such that σ is a free adjustable parameter; however, the implementation of the SVM (LIBSVM [21]) configures this number to *1*. The result is used to calculate the weight (*w*) and bias (*b*) in the form of Lagrange multipliers (line 2). Similar steps are applied for testing; however, only the test's individual input (*x*) (here, we only feed one line as input) will be applied to the kernel function over the training sets (*X*). Equation (4) will subsequently be applied to compute the objective function (*D*) and to determine the *CA*; here, *1* represents cacheable or else (Algorithm 6).

Algorithm 5: SVM (Classification) in the Training Phase

Input: $X(R, U, S)_N$, Y_N

Output: w_m, b

1. Apply Kernel Function

$$g(X) = \exp(-\gamma \| X - X[j] \|^2) \mid j \in 1 \cdots N$$

2. Derive Weights (**w**) and Bias (*b*) based on Lagrange Multipliers

Algorithm 6: SVM (Classification) in the Testing Phase

Input: $x(R, U, S)$, w_m, b

Output: *CA*

1. Apply Kernel Function

$$g(X) = \exp(-\gamma \| X - X[j] \|^2) \mid j \in 1 \cdots N$$

2. Compute the objective function (*D*) from equation (4)
3. Calculate *CA* based on the objective function (*D*)

3 Literature Survey

Several previous studies have described derivatives of traditional caching schemes, e.g., FIFO, LRU, LFU, and GDS [22], for online caching. However, these schemes have a key limitation in their replacement precision, especially given diverse web content (dynamic) [10-11]. To address this limitation, soft-computing-based approaches, including web caching, have recently been used to heuristically seek the optimal solution with a timing trade-off.

Previously, soft computing was used as a selection procedure for objects in memory. For example, Khalid *et al.* [12, 14] proposed a selection method called KORA (Khalid Shadow Replacement Algorithm), which was a pioneering technique in the integration of NNs for enhancing cache replacement schemes.

Considering caching strategies, Cobb and ElAarag [23] improved KORA by introducing and enhancing Neural Network Proxy Cache Replacement (NNPCR) using a BPNN for web caching replacement based on a rating score of *0* to *1* using five main inputs: URL, frequency, size, timestamp, and number of requests. The results revealed a considerable improvement in performance compared to the traditional web caching policy.

However, a key limitation of the BPNN is its computational complexity, which is generally not suitable for a real-time caching. In addition to an NN, Calzarossa and Valli [16] proposed FL as a replacement scheme. Four main parameters (i.e., size,

timestamp, accumulative frequency, and response time) are transformed into fuzzy rules, including low, medium, high, and very high. Their results indicated an improvement in caching replacement efficiency, particularly with small caching storage.

Most of the techniques described above directly applied soft computing as the caching replacement policy; however, its key limitation is its high computation time, especially when properly used for on-line computation. Therefore, several recent proposals have described hybrid approaches that use both TCRP and soft computing [24].

Another soft computing class, EC, has been evaluated as a cache replacement scheme. For example, Vakali et al. [18] proposed a web cache replacement policy that uses a GA to determine the density of web objects, including the retrieval rate (the product of latency and bandwidth), and then uses the density to construct the replacement rule. The results revealed a higher Hit Rate and Byte Hit Rate (HR and BHR) than other TCRP methods.

Sulaiman et al. [25] applied Particle Swarm Optimization (PSO) instead of a GA. The computation of PSO was used to analyze the distribution of web objects before feeding it into LFU for final replacement. The precision and time complexities of this method were superior to those obtained using a BPNN. In addition, recently, Samuel et al. [26] improved a traditional Naïve Bayes (NB) technique with multiple nodes to construct the tree and reported superior performance with GDS hybridized with frequency; however, with large cache size, the precision tends to be reduced.

Note that the hybrid models discussed above were integrated into a single storage, resulting in a key limitation on model interruption between TCRP and soft-computing-based approaches. Thus, there have also been some approaches addressing the concept of caching separation such as online and offline caches.

Ahmed and Shamsuddin [27] investigated a combination of these two approaches, called a Neuro-Fuzzy System (ANFIS), to predict whether a web object will be re-accessed within a given time period. In this approach, the caching schemes are either short term or long term (similar to online and offline caches). The first caching scheme applies LRU for fast retrieval, and the second scheme uses ANFIS. The four parameters, URL, time, frequency, and size, were also used as long-term cache parameters.

Similar to Samuel et al. [26], however, Ali et al. [28] also proposed NB for classification using a Bayesian network to identify two classes: high probability to use or not use again. Again, two components, similar to the long-term and short-term caches that were proposed by the same group of authors [27], were designed and found to achieve better performance.

Note that the same group of authors also considered an SVM instead of NB for classification [29]. However, here, GDSF was selected for the second component. The results showed an improvement in the efficacy of the replacement gain. To further reduce the computational time complexity, Sajeeva and Sebastian [30] applied the semi-intelligent concept of using Logistic Regression (LR) hybridized with LRU to improve the performance (computational time speed-up) but with a precision trade-off. Note that the concepts of two-component caching were not discussed in detail, and the use of the selection criteria instead of other criteria was not justified.

4 Two-Level Intelligent Web Caching Schemes

Considering the key advantages of TCRP and soft computing (i.e., fast computation and high precision), our approach integrates both techniques into a hybrid model. TCRP is first used for online caching, and the accuracy is increased using soft computing. The selection of LFU was based on its superior performance in our evaluation, and ELM was used to represent the soft-computing schemes due to its advantage of providing computational complexity reductions with increased precision. In this section, we provide a detailed description of our proposed method, which is called Two-Level Intelligent Web Caching Schemes Using Hybrid ELM-LFU (H-ELM-LFU). Figure 10 presents a general schematic of our approach, which can be divided into first- and second-level caching schemes.

4.1 First-Level Cache

In this level, the key engine performs the replacement using the traditional web caching scheme (TCRP) for the purpose of (fast) online caching. Here, the LFU was selected based on its results compared to FIFO, LRU, and GDS; however, it is not limited to this method. There are three main components as follows: the TCRP cache engine, LFU cache replacement, and Tree Removal.

TCRP cache engine. This component is the key function that responds to the user request. It has two main goals. First, if the web request exists in the first-level cache database (TRCP caching storage), this module immediately responds with the web object (*Cache Hit*). In case there is no such object, this module continues to make a request for the object to the second-level cache management (i.e., ELM cache engine, discussed later) from the larger database. If found, the web object will be returned to the user; in addition, it will be stored/updated in the first-level cache (also defined as *Cache Miss*).

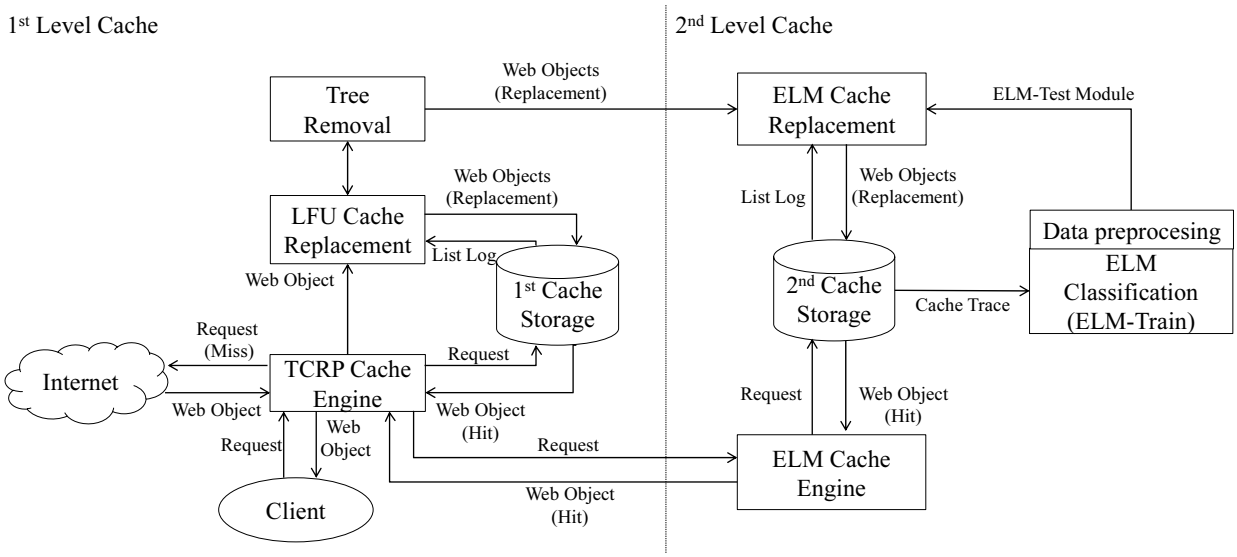


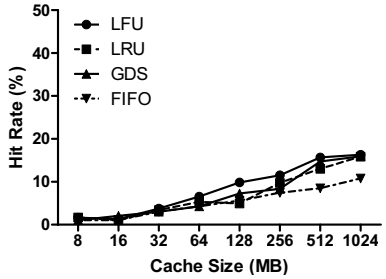
Figure 10. System overview of two-level intelligent web caching schemes

Second, if there are no such web objects, then this module will make a request to the web server (Internet) for that particular web object. Once it is received, this module will forward it to the user and simultaneously replicate the web object to store in the first-level storage (if it is not full), also defined as *Cache Miss*.

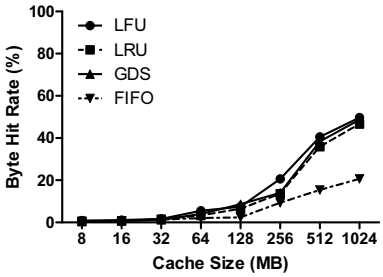
LFU cache replacement. This module will be activated only in the case of the cache being full (needed for replacement). This activation is used to identify the least frequently accessed web object in storage to be replaced with a new web object through the TCRP cache engine. Once replaced, a particular replacement web object will be forwarded to the second-level cache management, including the access pattern (hit/miss), which is the ELM cache replacement (discussed in detail in Section 4.2).

Note that, again, the selection of LFU is used for a fast caching replacement and particularly for real-time communication because of its key advantage of low complexity, i.e., $O(log(n))$ [31]. In general, LFU assumes that, the higher the probability of accessible web objects, the higher the probability for the object being cached, thereby reducing the latency of subsequent requests.

To confirm our selection criteria, we performed an intensive evaluation of well-known TCRPs, including FIFO, LRU, LFU, and GDS, and selected the best (LFU) for the web caching replacement in our first-level cache. Figure 11 shows the replacement performance of different algorithms using the BO2 web dataset from IRCache [32] over 15 days from the fourth quarter of 2015. We measured the HR and BHR under different cache sizes (8 MB to 1024 MB). The results clearly indicate the outstanding LFU performance of greater than 16% and of nearly 50% for the HR and BHR, respectively.



(a) % Hit Rate



(b) % Byte Hit Rate

Figure 11. Web caching performance using the BO2 dataset

Tree removal. The purpose of the TCRP tree removal is similar to that of cache replacement; however, this module facilitates LFU replacement, particularly for a related web object. According to previous research on hierarchical tree structures, each website generally consists of various web objects (e.g., HTML, PHP, style sheets, images, audio, and video), all of which will be stored in the cache to provide fast accessibility. Thus, a traditional caching scheme will select the replacement policy if and only if there is a target replacement; therefore, other related objects are likely still in storage and not used.

Although the objects will ultimately be chosen for replacement based on the frequency criteria, it may

take a longer period of time, and other, more significant objects will likely be replaced instead. Consequently, this study also proposes a specific rule to seek out related web objects. As shown in Figure 12, assume that the replaced web object *www.kku.ac.th* is considered to be the root (target replacement). Once it is replaced, other related objects, called leafs, will also be removed; in this case, these objects are all the objects sub-directories such as */vendor* and */images*.

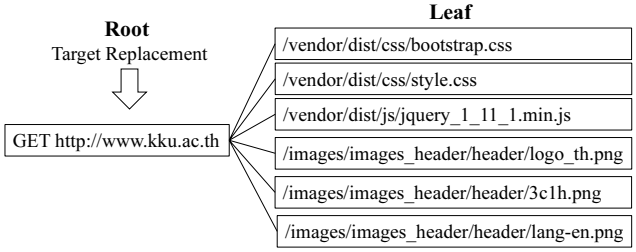


Figure 12. Example of tree removal

4.2 Second-Level Cache

The main purpose of this level is to achieve high accuracy (i.e., high HR and/or BHR) with the support of larger storage. Thus, we propose the use of an intelligent caching algorithm that uses one of the soft computing techniques (ELM) and its enhancement for classification. There are four components: the ELM cache engine, ELM cache replacement, data pre-processing, and ELM classification.

ELM cache engine. This module is mainly used to respond to the web object request from the first-level cache (if missed) by seeking the object in the second-level storage (ELM caching storage). This module also returns "miss" if no such object is available; however, the actual object will be returned if found.

ELM cache replacement. This component functions together with the latter two modules of the first-level cache. The web objects that are chosen to be replaced in the first-level storage will be used as inputs to replace objects in the second-level storage. If the storage is not full, the web objects will be directly stored.

However, if the storage is full, this module will perform ELM testing to determine if the objects should be stored based on the ELM training model. If the result is "cacheable" or "should be cached", this module will perform the actual cache replacement procedure in sequence; otherwise, the particular object will be ignored.

To improve the model precision, after testing, the "cacheable" and "uncacheable" statuses will both be used to update the re-training process, which will be recomputed based on a specific threshold (the cache storage size). Here, we used 100%, i.e., all objects in storage will be replaced.

Data pre-processing. This component is used to prepare the web object in storage for the next step, which is to apply the optimized ELM classification scheme. In general, the web objects will be converted into a particular format. Here, the characteristics of the web query, i.e., HTTP request method (*R*), URL_ID (*U*), size (*S*), and HTTP code (hit or miss), are used for the transformation into a particular format in the range between *0* and *1*.

ELM classification. Once the data, which are a representation of web objects, such as the web access log, are ready from the previous step, this module is mainly used to construct the replacement decision criteria (ELM model) to make a final decision about which of the objects should be cached or replaced. Because various web objects are periodically updated, ELM also requires an update of the training to identify the caching condition. Note that only a log structure (cache trace) will be used for classification to avoid cache storage intervention.

4.3 Intelligent Web Caching Schemes

This section describes ELM classification techniques, particularly in the context of web caching and its enhancement. These approaches are used to generate the ELM model for testing new web objects.

Extreme learning machine (ELM). Huang et al. [33] first proposed the ELM. The ELM applies a single-hidden-layer feed-forward network (SLFN), and thus, there is no requirement to adjust the hidden node weight as in an NN. The key advantage of the ELM is that it is a fast training algorithm.

Figure 13 presents a schematic of the ELM. Given a total of N input neurons in the format (x_i, t_i) such that $i = 1, 2, 3, \ldots, N$, the input $x_i = (x_{i1}, x_{i2}, \ldots, x_{iN})^T$ is input into the network considering the target $t_i = (t_{i1}, t_{i2}, \ldots, t_{im})^T$. The output weight β and the bias (b) are used to convert the non-linearity and are derived from the equation below (for training purposes).

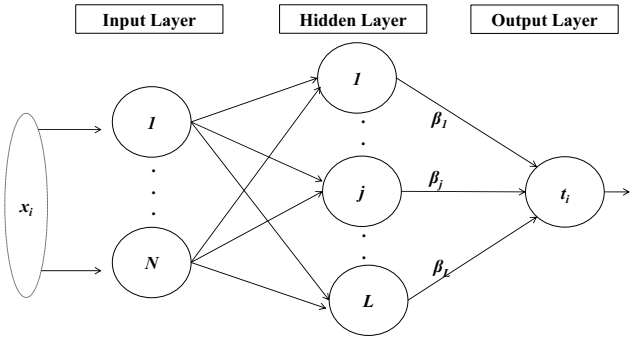


Figure 13. ELM (SLFN)

$$\beta = H^{\dagger}T \tag{5}$$

The output weight β can be obtained by resolving the least-squares solution as stated in equation (5). Here, T denotes the target $[t_1, t_2, \ldots, t_N]^T$ and H is a hidden node function, such as $\{h_{ij}\} \mid (i = 1, \ldots, N$ and $j = 1, \ldots, L)$, which is derived from $h_{ij} = g(w_j x + b)$. $H^{\dagger}$ is a Moore-Penrose matrix, where the hidden node

input weights are $w_j = [w_{j1}, w_{j2}, ..., w_{jN}]^T$ with a bias of b_j, and $g()$ is the activation function, i.e., hard limit, sigmoid, radial basis, triangular basis, or sine.

During the testing stage, the unknown input x will be used in $h_{ij} = g(w_j x + b)$, and the defined weight (w) will be given before applying the activation function. The reverse equation below is then used to compute the predicted target as shown in equations (6) to (8).

$$\min \sum_{i=1}^{N} \| \beta_i \cdot h_i - t_i \| \quad \textbf{(6)}$$

where $T = H\beta$

$$H(w_1,\cdots,w_L,b_1,\cdots,b_L,x_1,\cdots,x_N) = \begin{bmatrix} g(w_1 \cdot x_1 + b_1) & \cdots & g(w_L \cdot x_1 + b_L) \\ \vdots & & \vdots \\ g(w_1 \cdot x_N + b_1) & \cdots & g(w_L \cdot x_N + L) \end{bmatrix} N \times L \quad \textbf{(7)}$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_L^T \end{bmatrix} \text{and } T = \begin{bmatrix} T_1^T \\ \vdots \\ T_L^T \end{bmatrix} \quad \textbf{(8)}$$

ELM for web caching replacement. To apply the ELM for web caching replacement, three attributes are used as the main features (R, U, S), including the status of the cache T (hit or miss). Algorithm 7 presents the detailed methodology. First, these X inputs are transformed to the range of [0, 1] (line 1), and the input weight (w) and the bias (b) are randomly generated (lines 2-3). The activation function (H) is subsequently applied (line 4), and the output weight (β) is derived accordingly (line 5). For testing, algorithm 8 shows that, following input transformation, processes similar to those of training will be performed to determine the cacheable probability (CA) by solving equation (7).

Algorithm 7: ELM (Classification) in the Training Phase

Input: $X(R, U, S, T)_N$, T_N

Output: $w_{K,N}, \beta_K, B_{K,N}$

1. Normalize input parameters X in the range [0, 1]
2. Generate random input weight $w_{K,N}$ in the range [-1, 1]
3. Generate random bias $b_{K,N}$ in the range [0, 1]
4. Calculate Activation Function
$$H = G(w_{K,N} \times X(R,U,S) + b_{K,N})$$
5. Calculate the output weight
$$\beta_K = H^{\dagger} \times T_N$$

Algorithm 8: ELM (Classification) in the Testing Phase

Input: $x, w_{K,N}, \beta_K, b_{K,N}$

Output: CA

1. Normalize input parameters x in the range [-1, 1]
2. Calculate Activation Function
$$H\_Test = G(w_{K,N} \times x(R,U,S) + b_{K,N})$$
3. Compute Cacheable Probability
$$CA = (H\_Test)^T \times \beta_K$$

ELM with similarity feature. Although several activation functions affect the replacement precision, our evaluation (See also Section 5) showed that the ELM with a hard limit outperforms the NN, FL, GA, and SVM. Thus, the ELM with a hard limit was selected for further optimization. This research also proposes an additional feature to improve the classification precision using the similarity factor or a factor to differentiate the two objects. The rationale behind this feature is two-fold.

The first is for in the case that the actual object is available. Here, the concept of the hashing sum is used to identify the object similarity even though the actual URL may be different. We applied "md5sum" [34] to generate a fixed size string to represent the whole object. The example is as follows: Input = "$md5sum nav_logo242.png" and Output = "710544E7F0C828 B42F51207342622D33."

However, there is a computational time trade-off relative to the object size because the hash of the entire object will be computed. Thus, second, to speed-up the time complexity as well as in the case of the unavailability of a web object, here, we propose the use of a similarity factor (SF), a combination of different factors from the only trace structure; this combination, called ELM with similarity, is shown in equation (9).

$$SF = \frac{(\frac{LF}{Size} + CType)}{(LF_{Max} + CType_{Max})} \quad \textbf{(9)}$$

Where SF denotes the similarity factor, with the summation of LF over $Size$ and $CType$; LF is the total length of a specific file name, which corresponds to its size ($Size$); and $CType$ is the content type with its encoding format (i.e., application = 1, audio = 2, binary = 3, font = 4, image = 5, text = 6, video = 7, and other = 8). This summation is also normalized by the maximum values of these two factors, i.e., LF_{Max} (255) and $CType_{Max}$ (8).

5 Performance Evaluation

In this section, the performance and practicality of our proposed techniques are evaluated and compared with other existing candidates [14, 17-18, 29] for the integration of the traditional caching policy and

intelligent systems (NN, FL, GA, and SVM), including ELM optimization as well as LFU and LRU as a representation of the TCRP.

5.1 Data Pre-processing

We selected a well-known real-world web cache dataset, IRCache [32], from the proxy log from the National Lab of Applied Network Research (NLANR), including the access trace from five main proxy servers across the U.S. (i.e., UC, BO2, SD, SV, and NY). Note that most related evaluations of web caching have also used these traces [17-18, 20, 26-30]. The implementation of the algorithm into the actual proxy, including the proxy placement into the commercial (operation) network, is a future consideration due to management and administrative policy constraints.

However, these traces reflect the actual Internet usage across the U.S. Because the actual trace includes millions of objects, to compare the technique with other candidates [20, 26-30], the trace was limited to 15 days from the fourth quarter of 2015. Note that the size of the cache is overwhelmed by the number of web objects. Table 2 shows a detailed dataset from IRCache.

Table 2. Web caching dataset (IRCache)

Proxy Dataset	Proxy Server	Location	#Records	Duration (days)
UC	uc.us.ircache.net	Urbana–Champaign, IL	1,548,547	15
BO2	bo.us.ircache.net	Boulder, CO	1,357,461	15
SD	sd.us.ircache.net	Silicon Valley, CA	1,249,572	15
SV	sv.us.ircache.net	San Diego, CA	1,189,115	15
NY	ny.us.ircache.net	New York, NY	1,587,544	15

The trace files acquired from the IRCache record all requested web information that was used to make a decision such as web cache replacement. The files include seven main attributes: the timestamp (with socket status as closed) in milliseconds (ms), the client address (IP address of the requester to the proxy server), the tag and HTTP code (the status of accessible codes, i.e., hit or miss), the size of the web object in bytes, the request method (e.g., GET, POST, or PUT), the URL, and the content type (such as html, video, or audio).

5.2 Performance Measurement Metric

There are two main metrics for the performance of the web caching scheme used in this study.

Classification metric. This metric is mainly used to state the classification precision of the intelligent system or if the particular web object should be cached. Here, the metric is the Corrected Classification Rate (CCR) [9], as given by equation (10) below.

$$CCR = \frac{TP + TN}{TP + FP + FN + TN}(\%) \quad \textbf{(10)}$$

Where *TP* (True Positive) is the classification result whereby positive training data are evaluated as positive, *TN* (True Negative) denotes the classification result whereby negative training data are evaluated as negative, *FP* (False Positive) is the classification result whereby negative training data are evaluated as positive, and *FN* (False Negative) denotes the classification result whereby positive training data are evaluated as negative.

Two-level web cache metric. Two well-known metrics, HR and BHR, are commonly used to determine the overall performance of web systems. The HR is the ratio of the number of web objects that the proxy server can deliver directly back to the client, and its corresponding size is denoted as BHR [7, 8, 13, 26-30].

In addition, we measured the computation time during the evaluation. To reflect the overall performance, we performed both unit and system testing. We measured the computational complexity of both the classification analysis and the two-level web caching.

5.3 Simulation Setups and Configurations

There are two metrics used for the evaluation process, including the computation time measurement. Thus, there are also two main configurations, which are stated below.

Classification configuration. Our proposal is based on ELM classification optimization, which was originally obtained from Huang *et al.* [33] using the MATLAB tool. There are two main scenarios:

Scenario 1. The following four well-known soft computing techniques that are applied to web cache replacement policies [7] are used to evaluate the classification precision: NN [12], FL [16], GA [18], and SVM (RBF) [29]. RBF was selected due to its superior performance among the different kernel functions including time complexity measurements (training and testing). The parameters related to the intelligent scheme include configurations that follow the recommendation from the literature [7, 12, 16, 18, 29].

For example, the NN applies a BPNN with a range of weights in [-1, 1] [12]. Four main variables (i.e., time stamp, size, accumulative frequency, and response time), including 12 fuzzy rules, were used [16]. Similarly, three of the variables, excluding the accumulative frequency, were used to compute the fitness for the GA [18].

For this scenario, the dataset was acquired from SD, with 1,249,572 transactions over 15 days. The evaluation applied *K*-fold cross validation [9], in which *K* is set to 4 [20, 23, 27, 30] (i.e., 75% for training and 25% for testing) for four rounds. We limited the maximum cache size to 1 GB to be consistent with the setups in scenario 2 and 3.

Scenario 2. We evaluated the practical use of our proposed technique using the ELM [33] under various

activation functions, i.e., sine, sigmoid, hard limit, triangular basis, and radial basis functions, using CCR and again with time measurements. The same dataset used in Scenario 1 was applied to compare the performance.

Two-level web cache configuration. With the results from the first two scenarios, this scenario (**Scenario 3**) was evaluated to determine the actual caching efficiency in terms of HR and BHR. We adapted other comparative hybrid schemes to evaluate the performance, one of which was proposed by Ali *et al.* [29], who integrated LRU with an SVM (here called SVM-LRU). Similarly, the other scheme was proposed by Sathiyamoorthi and Ramya [35] but represents the integration of an SVM and LFU (here called SVM-LFU).

As discussed in Section 2, LRU and LFU were used for comparison. The emulation is based on WebTraff, which is a module for evaluating caching efficiency with real-world traces. It was installed on a system running Linux Ubuntu 12.04 LTS and with a 2.66 GHz Intel(R) Core (TM) Quad Q8400 CPU, 4 GB DDR-SDRAM, and a 250 GB, 5400 rpm hard disk. Five main web access datasets, with cache sizes of 2^k ranging from 3 to 10, were evaluated.

5.4 Simulation Results and Discussion

Three main scenarios are discussed here. To compare the performance of the soft-computing-based classifications, i.e., NN, FL, GA, and SVM, for the first scenario, Table 3 shows the CCRs of the different soft computing approaches; higher scores indicate better performance. The classification performance of the NN is outstanding (94.90%) compared with the approximately 85% performance of the FL and GA; however, the accuracy of the SVM, 93.44%, is not significantly different from that of the NN.

Table 3. CCRs of Soft computing approaches

Soft Computing	CCR	Time (ms)
NN	94.90	4,580
FL	83.86	430
GA	85.38	650
SVM	93.44	1,125

Considering the computation time complexity, as is generally known, the precision of the NN is the best but is achieved at a high computation time trade-off, i.e., 4,580 ms. In contrast, the FL is outstanding in terms of performance (only 430 ms) but suffers from a precision trade-off. Similarly, the computation time of the GA is almost eight times lower than that of the NN. The SVM requires approximately 1 second, which is between the times of FL/GA and the NN (but again with high precision).

Table 4 compares the performance (CCR) of the ELM classification with various activation functions (i.e., sine, sigmoid, hard limit, TRIBAS, and RBF) in Scenario 2. In general, the ELM (all functions) can achieve a high classification rate (greater than 94%), although the ELM with the hard limit function is outstanding (95.15%), and the other four methods are similar (average of approximately 94.88%).

Table 4. CCRs of ELM (Various activation functions)

ELM (Activation Function)	CCR	Time (ms)
Sine	94.85	1,211
Sigmoid	94.85	1,280
Hard limit (HARD)	95.15	1,274
Triangular basis (TRIBAS)	94.80	1,211
Radial basis (RBF)	94.77	1,291

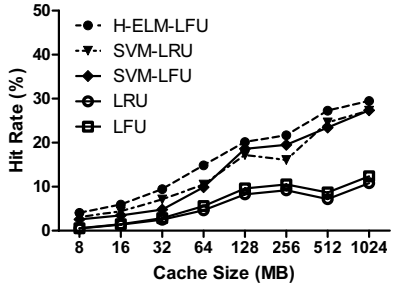
The computational times of all of the functions are similar (between 1,211 ms and 1,291 ms), but that of the RBF is the worst. The sine and TRIBAS functions obtained the best performance (1,211 ms) but at lower CCR compared with the ELM with HARD, which was then used for our subsequent experiment due to its superior classification rate and time complexity trade-off (found not to be significantly different from the other functions).

Comparison of Table 3 and Table 4 shows that the ELM (hard limit) achieves the highest recognition rate (95.15%); thus, it was again selected for further ELM optimization.

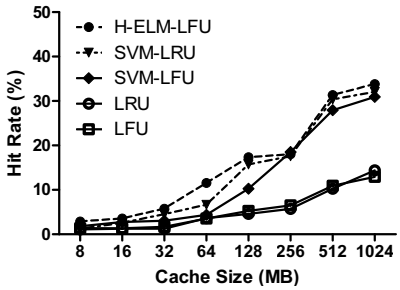
Figure 14 and Figure 15 compare the performance of the SVM with LRU [29] and LFU [35] for Scenario 3, including our enhancement (H-ELM-LFU) as well as the traditional LRU and LFU. In general, the HR trend of the five datasets is similar: the larger the cache size, the higher the HR (see Figure 14). This relationship is reasonable because there is a large opportunity for web objects to be available in the cache.

Given a particular cache replacement policy, the performance of H-ELM-LFU is generally outstanding; it ranges from approximately 3% to 38% for cache sizes of 8 MB to 1024 MB. The performances of the other classifications descend in the order of SVM-LRU, SVM-LFU, LFU, and LRU. The performances of the last two approaches are similar (due to the use of only traditional caching).

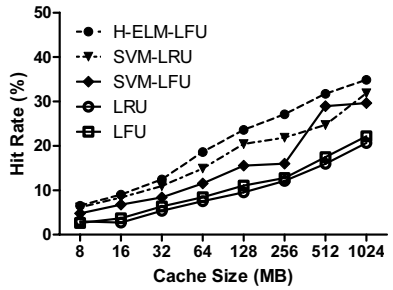
Although there is an obvious effect from the hybrid model, H-ELM-LFU still maintains an outstanding HR (17.96% on average); its average performance improvements on five datasets over SVM-LRU and SVM-LFU are 21.3% and 24.6%, respectively. In other words, the average HRs of these two methods are only 14.81% and 14.41%. In addition, LFU is superior to LRU; the HRs are approximately 9.23% and 9.04%, respectively. Compared with H-ELM-LFU, the percentage improvements are over 94.5% and 98.7%, respectively.



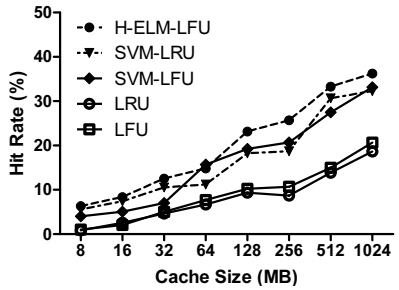
(a) UC dataset



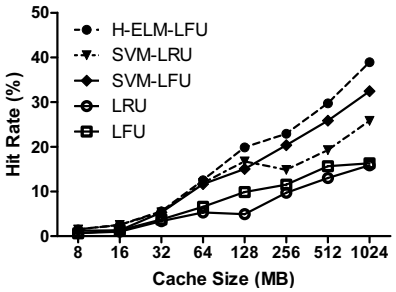
(b) SV dataset



(c) SD dataset

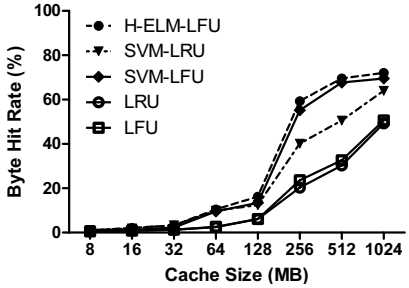


(d) NY dataset

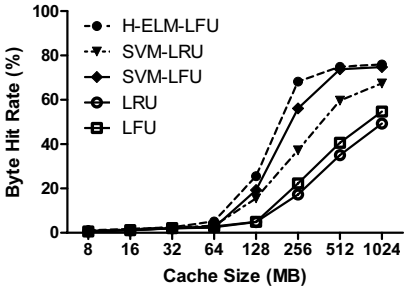


(e) BO2 dataset

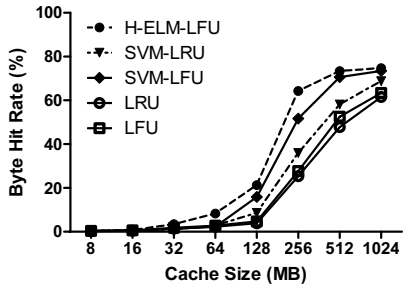
Figure 14. % Hit rate vs. cache size (MB)



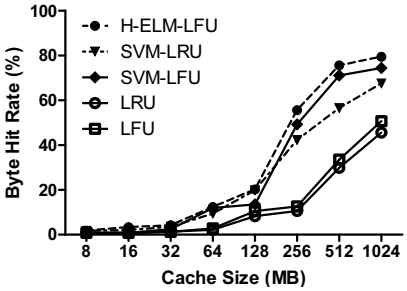
(a) UC dataset



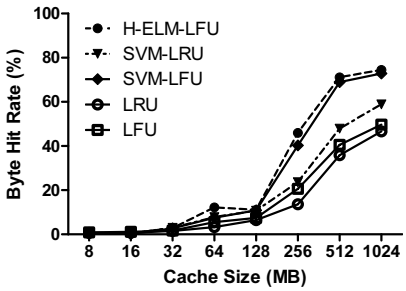
(b) SV dataset



(c) SD dataset



(d) NY dataset



(e) BO2 dataset

Figure 15. % Byte hit rate vs. cache size (MB)

Figure 15 shows the BHR values that correspond to the HR values in Figure 14 but with different orders (here, SVM-LFU is better than SVM-LRU). The trend closely follows that of the HR; the larger the cache size, the higher the BHR. The performance of H-ELM-LFU remains outstanding, namely, up to 79%. LFU and LRU more strongly affect the BHR than the HR (up to 50% and 45%).

The performances of the two hybrid approaches are similar (SVM-based approaches) and are also similar to H-ELM-LFU. However, the performance of H-ELM-LFU is again outstanding (61% on average) and in the order of SVM-LFU, SVM-LRU, LFU, and LRU. The average performance improvements of H-ELM-LFU over the other four methods are 21.1%, 33.65%, 87.87%, and 94.8%, respectively.

In addition, Figure 16 shows the computation time performance. Here, we measured over the entire caching epoch and then performed the average over five datasets. In general, with larger cache size, the computation time is reduced due to the additional space available to perform the replacement. The timing ranges from approximately 200 to 40 seconds with 8 MB to 1024 MB cache sizes. The computation times for all techniques (using traditional caching or the hybrid model) are not significantly different due to the parallel processing of both caching levels.

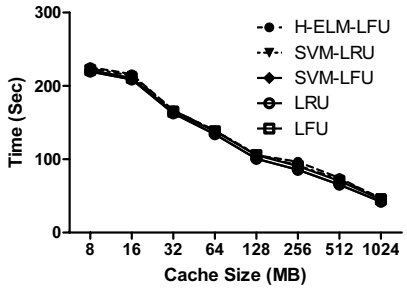


Figure 16. Web caching performance: computation time

6 Conclusions and Future Work

This study evaluated a two-level web caching system that was designed to achieve high HR and BHR with a resource constraint (i.e., caching storage), including support for real-time caching. The fast traditional caching policy was used for the front end, and an intelligent system was applied to create caching opportunity information for the back end to enhance the replacement precision.

Several traditional caching policies (i.e., FIFO, LRU, LFU, and GDS) were also evaluated, and the results demonstrated the outstanding performance of LFU. Thus, LFU was selected for the front end. Several soft computing approaches, including NN, FL, GA, and SVM, were also examined for the second-level cache. Our proposed method, which is called ELM optimization (ELM Similarity with a Hard Limit), and the seamless integration with the traditional caching policy (LFU) were studied (Hybrid ELM-LFU or H-ELM-LFU) and found to provide an HR of nearly 38% and a BHR of 79%.

Although our study indicates that H-ELM-LFU performance is a superior web caching replacement system, additional investigations, assumptions, and constraints, such as heterogeneous caching access and large-scale datasets that include large-scale caching, should be explored. Other hybrid schemes and optimizations of soft computing should also be investigated in terms of time complexity trade-offs. Due to the limitation of proxy placement access by the Internet Provider, practical implementation of H-ELM-LFU should also be investigated further. These topics are all subjects of ongoing research.

Acknowledgement

This research was supported by a grant from the Faculty of Science, Khon Kaen University and Khon Kaen University.

References

[1] Internet Society, *Internet Society Global Internet Report*, Report 2017, January, 2018.

[2] Cisco, *Cisco Visual Networking Index: Forecast and Methodology 2016-2021*, Report 2017, June, 2017.

[3] w3schools.com, *Browser Statistics and Trends*, https://www.w3schools.com/BROWSERS/default.asp.

[4] D. Tuncer, V. Sourlas, M. Charalambides, M. Claeys, J. Famaey, Scalable Cache Management for ISP-Operated Content Delivery Services, *IEEE Journal on Selected Areas in Communications*, Vol. 34, No. 8, pp. 2063-2076, August, 2016.

[5] C. Imbrenda, L. Muscariello, D. Rossi, Analyzing Cacheable Traffic in ISP Access Networks for Micro-CDN Applications via Content-Centric Networking, *Proceedings of the First International Conference on Information-Centric Networking*, Paris, France, September, 2014, pp. 57-66.

[6] S. Mittal, J. S. Vetter, A Survey Of Architectural Approaches for Data Compression in Cache and Main Memory Systems, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 27, No. 5, pp. 1524-1536, May, 2016.

[7] W. Ali, S. M. Shamsuddin, A. S. Ismail, A Survey of Web Caching and Prefetching, *International Journal of Advances in Soft Computing and its Application*, Vol. 3, No. 1, pp. 1-27, March, 2011.

[8] S. Sulaiman, S. M. Shamsuddin, A. Abraham, A Survey of Web Caching Architectures or Deployment Schemes, *International Journal of Innovative Computing*, Vol. 3, No. 1, pp. 5-14, June, 2013.

[9] H. Jang, E. Topal, A Review of Soft Computing Technology Applications in Several Mining Problems, *Applied Soft Computing*, Vol. 22, pp. 638-651, September, 2014.

[10] G. Zhang, Y. Li, T. Lin, Caching in Information Centric

Networking: A Survey, *Computer Networks*, Vol. 57, No. 16, pp. 3128-3141, November, 2013.

[11] L. A. Belady, R. A. Nelson, G. S. Shedler, An Anomaly in Space-time Characteristics of Certain Programs Running in a Paging Machine, *Communications of the ACM*, Vol. 12, No. 6, pp. 349-353, June, 1969.

[12] H. Khalid, A New Cache Replacement Scheme Based on Backpropagation Neural Networks, *ACM SIGARCH Computer Architecture News*, Vol. 25, No. 1, pp. 27-33, March, 1997.

[13] W. Ali, S. Sulaiman, N. Ahmad, Performance Improvement of Least-Recently-Used Policy in Web Proxy Cache Replacement Using Supervised Machine Learning, *International Journal of Advances in Soft Computing and its Applications*, Vol. 6, No. 1, pp. 1-38, March, 2014.

[14] H. Khalid, Performance of the KORA-2 Cache Replacement Scheme, *ACM SIGARCH Computer Architecture News*, Vol. 25, No. 4, pp. 17-21, September, 1997.

[15] G. G. Vijayan, J. S. Jayasudha, A Survey on Web Pre-fetching and Web Caching Techniques in a Mobile Environment, *Computer Science & Information Technology*, Vol. 2, No. 1, pp. 119-136, January, 2012.

[16] M. C. Calzarossa, G. Valli, A Fuzzy Algorithm for Web Caching, *Simulation Series Journal*, Vol. 35, No. 4, pp. 630-636, 2003.

[17] M. M. Bartere, P. V. Ingole, A Survey on Applications of Genetic Algorithms and Fuzzy Logic in Caching, *International Journal of Science and Engineering Investigations*, Vol. 1, No. 2, pp. 5-7, March, 2012.

[18] A. Vakali, Evolutionary Techniques for Web Caching, *Distributed and Parallel Databases*, Vol. 11, No. 1, pp. 93-116, January, 2002.

[19] DomainTools, *Domain Count Statistics for TLDs*, http://www.research.domaintools.com/statistics/tld-counts/.

[20] W. Ali, S. M. Shamsuddin, A. S. Ismail, Intelligent Web Proxy Caching Approaches Based on Machine Learning Techniques, *Decision Support Systems*, Vol. 53, No. 3, pp. 565-579, June, 2012.

[21] C. Chang, C. Lin, LIBSVM: A Library for Support Vector Machines, *ACM Transactions on Intelligent Systems and Technology*, Vol. 2, No. 3, pp. 1-27, April, 2011.

[22] M. Dawar, C. Singh, Study on Web Caching Architecture: A Survey, *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol. 3, No. 11, pp. 581-585, November, 2013.

[23] J. Cobb, H. E. Aarag, Web Proxy Cache Replacement Scheme Based on Back-Propagation Neural Network, *Journal of Systems and Software*, Vol. 81, No. 9, pp. 1539-1558, September, 2008.

[24] A. Abdalla, S. Sulaiman, W. Ali, Intelligent Web Objects Prediction Approach in Web Proxy Cache Using Supervised Machine Learning and Feature Selection, *International Journal of Advances in Soft Computing and Its Applications*, Vol. 7, No. 3, pp. 146-164, November, 2015.

[25] S. Sulaiman, S. M. Shamsuddin, F. Forkan, A. Abraham, Intelligent Web Caching Using Neurocomputing and Particle Swarm Optimization Algorithm, *Proceedings of the Asia International Conference on Modelling & Simulation (AICMS)*, Kuala Lumpur, Malaysia, 2008, pp. 642-647.

[26] P. J. Benadit, F. S. Francis, U. Muruganantham, Improving the Performance of a Proxy Cache Using Tree Augmented Naive Bayes Classifier, *Procedia Computer Science*, Vol. 46, pp. 184-193, May, 2015.

[27] W. A. Ahmed, S. M. Shamsuddin, Neuro-Fuzzy System in Partitioned Client-Side Web Cache, *Expert Systems with Applications*, Vol. 38, No. 12, pp. 14715-14725, November-December, 2011.

[28] W. Ali, S. M. Shamsuddin, A. S. Ismail, Intelligent Naïve Bayes-based Approaches for Web Proxy Caching, *Knowledge-Based Systems*, Vol. 31, pp. 162-175, July, 2012.

[29] W. Ali, S. M. Shamsuddin, A. S. Ismail, Intelligent Web Proxy Caching Approaches Based on Support Vector Machine, *International Conference on Informatics Engineering and Information Science (ICIEIS 2011)*, Vol. 252, Kuala Lumpur, Malaysia, 2011, pp. 559-572.

[30] G. P. Sajeev, M. P. Sebastian, Building Semi-Intelligent Web Cache Systems with Lightweight Machine Learning Techniques, *Computers & Electrical Engineering*, Vol. 39, No. 4, pp. 1174-1191, May, 2013.

[31] Z. Shuchang, An Efficient Simulation Algorithm for Cache of Random Replacement Policy, *IFIP International Conference on Network and Parallel Computing (NPC)*, Zhengzhou, China, 2010, pp. 144-154.

[32] National Lab of Applied Network Research (NLANR), *Sanitized Access Logs*, http://www.ircache.net/.

[33] G. Huang, D. H. Wang, Y. Lan, Extreme Learning Machines: A Survey, *International Journal of Machine Learning and Cybernetics*, Vol. 2, No. 2, pp. 107-122, June, 2011.

[34] Unix.com, *Man Page for md5sum*, https://www.unix.com/man-page/linux/1/md5sum.

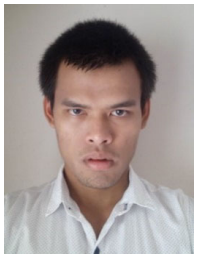
[35] V. Sathiyamoorthi, P. Ramya, Enhancing Proxy Based Web Caching System using Clustering Based Pre Fetching with Machine Learning Technique, *International Journal of Research in Engineering and Technology*, Vol. 3, No. 7, pp. 463-469, May, 2014.

Biographies

Phet Imtongkhum is an M.S. student in the Department of Computer Science, Khon Kaen University, Thailand, where he also received a BS degree (honor) in 2013. He is CEO of Advanced Internetworking Co. Ltd. His research interests include computer networking, multimedia networks, and machine learning.

Chakchai So-In (IEEE/ACM SMs) is a Professor (Associate) in the Department of Computer Science at Khon Kaen University. He received PhD from WUSTL, USA. He was an intern at Cisco Systems, WiMAX Forum, and Bell Lab. He has published over 80 papers and has authored 10 books. His research interests include mobile, sensor, wireless, and computer networks.

Surasak Sanguanpong is an Associate Professor in the Department of Computer Engineering and the Director of the Applied Network Research Laboratory at Kasetsart University. He received a B.Eng. and an M.Eng. in Electrical Engineering from Kasetsart University in 1985 and 1987. His researches focus on network measurement, Internet security and high-speed networking.



Songyut Phoemphon is a Ph.D. student in the Department of Computer Science, Khon Kaen University, Thailand, where he also received B.S. and M.S. degrees (honors) in 2014 and 2016. He was an intern at NECTEC, Thailand. His research interests include mobile computing, wireless sensor networks, and machine learning.