ISSN: 2277-9655
Impact Factor: 1.852

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

Novel Web Proxy Cache Replacement Algorithms using Machine Learning Techniques for Performance Enhancement

P. N. Vijaya Kumar[*1], Dr. V. Raghunatha Reddy[2]

[*1]PhD Research Scholar, Department of Computer Science & Technology, Sri Krishnadevaraya University, Anantapur-515003, India

[2]Assistant Professor, Department of Computer Science & Technology, Sri Krishnadevaraya University, Anantapur-51500, India

pnvijay.research@gmail.com

Abstract

A web cache is a mechanism for the temporary storage (caching) of web documents, such as HTML pages and images, to reduce bandwidth usage, server load, and perceived lag. A web cache stores copies of documents passing through it; subsequent requests may be satisfied from the cache if certain conditions are met. In this paper, machine learning techniques are used to increase the performances of traditional Web proxy caching policies such as SIZE, and Hybrid. Naïve Bayes (NB) and decision tree (C4.5) are used and integrated with traditional Web proxy caching techniques to form better caching approaches known as NB–SIZE, and C4.5–Hybrid. The proposed approaches are evaluated by trace-driven simulation and compared with traditional Web proxy caching techniques. Experimental results have revealed that the proposed NB–SIZE and C4.5–Hybrid significantly increased Pure Hit-Ratio, Byte Hit-Ratio and to reduced the latency when compared with SIZE and Hybrid.

Keywords: Web caching, Proxy Cache, Cache replacement, Classification, Naïve Bayes, Decision tree.

Introduction

A proxy server is a server (a computer system or an application) that acts as an intermediary for requests from clients seeking resources from other servers. A client connects to the proxy server, requesting some service, such as a file, connection, web page, or other resource available from a different server and the proxy server evaluates the request as a way to simplify and control its complexity. The efficiency of proxy caches is influenced to a significant extent by document placement/replacement algorithms. Web proxy caching plays a key role in improving Web performance by keeping Web objects that are likely to be visited again in the proxy server close to the user. This Web proxy caching helps in reducing user perceived latency, i.e. delay from the time a request is issued until response is received, reducing network bandwidth utilization, and alleviating loads on the original servers. Since the space apportioned to a cache is limited, the space must be utilized effectively. Therefore, an intelligent mechanism is required to manage Web cache content efficiently. The cache replacement is the core or heart of Web caching. Thus, the design of efficient cache replacement algorithms is extremely important and crucial for caching mechanism achievement [1]. The most common Web caching methods are not efficient enough and may suffer from a cache pollution problem, since they consider just one factor and ignore other factors that may have an impact on the efficiency of Web proxy caching [2]. Many Web proxy caching policies have attempted to combine some factors which can influence the performance of Web proxy caching for making decisions about caching. However, this is not an easy task, because one factor in a particular environment may be more important in other environments [3]. The challenge lies in predicting which Web objects should be cached and which 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 [4].

Web proxy log files record the activities of the users in a Web proxy server. These proxy log files contain complete and prior knowledge of future accesses. The availability of Web proxy log files that can be used as training data is the main motivation for utilizing machine learning techniques in adopting Web caching approaches. Recent studies have proposed exploiting machine learning techniques to cope with the above problem [5]. The Bayesian networks and decision tree (C4.5) are two popular supervised learning algorithms that perform classifications more accurately and faster than other algorithms [7, 8]. These machine learning algorithms have a wide range of applications such as text

ISSN: 2277-9655
Impact Factor: 1.852

classification, Web page classification and bioinformatics applications. Hence, Naïve Bayes classifier and C4.5 can be utilized to produce promising solutions for Web proxy caching. Naïve Bayes classifier is a simple Bayesian network classifier, which has been applied successfully in many domains.

This paper combines the most significant factors using common classifiers for predicting Web objects that can be re-visited later. In this paper, we present new approaches that depend on the capability of Naïve Bayes and C4.5 classifiers to learn from Web proxy logs files and predict the classes of objects to be re-visited or not. The trained Naïve Bayes and C4.5 classifiers can be effectively incorporated with traditional Web proxy caching algorithms to present novel Web proxy caching approaches with good performance in terms of hit ratio and byte hit ratio. The remaining parts of this paper are organized as follows. Background and related works are presented in Section 2. The framework for improved Web proxy caching approaches based on machine learning techniques is illustrated in Section 3. Implementation and experimental results are presented in Section 4. Section 5 discusses performance evaluation and discussion. Finally, Section 6 concludes the paper.

Background and Related Works

How Web Caches Work

Web caching is the temporary storage of web objects (such as HTML documents) for later retrieval. There are three significant advantages to web caching: reduced bandwidth consumption (fewer requests and responses that need to go over the network), reduced server load (fewer requests for a server to handle), and reduced latency (since responses for cached requests are available immediately, and closer to the client being served). Together, they make the web less expensive and better performing.

All caches have a set of rules that they use to determine when to serve an object from the cache, if it's available. Some of these rules are set in the protocols (HTTP 1.0 and 1.1), and some are set by the administrator of the cache (either the user of the browser cache, or the proxy administrator).

Generally speaking, these are the most common rules that are followed for a particular request (don't worry if you don't understand the details, it will be explained below):

1. If the object's headers tell the cache not to keep the object, it won't. Also, if no validator is present, most caches will mark the object as uncacheable.
2. If the object is authenticated or secure, it won't be cached.
3. A cached object is considered fresh (that is, able to be sent to a client without checking with the origin server) if:

- It has an expiry time or other age-controlling directive set, and is still within the fresh period.
- If a browser cache has already seen the object, and has been set to check once a session.
- If a proxy cache has seen the object recently, and it was modified relatively long ago.

Fresh documents are served directly from the cache, without checking with the origin server.

4. If an object is stale, the origin server will be asked to validate the object, or tell the cache whether the copy that it has is still good.

Together, freshness and validation are the most important ways that a cache works with content. A fresh object will be available instantly from the cache, while a validated object will avoid sending the entire object over again if it hasn't changed. Web content can be cached at a number of different locations along the path between a client and an origin server. The 3 types of Web Caches are Browser Cache, Proxy Cache, and Surrogate/Server Cache. The different types of caches are shown in Figure 1.

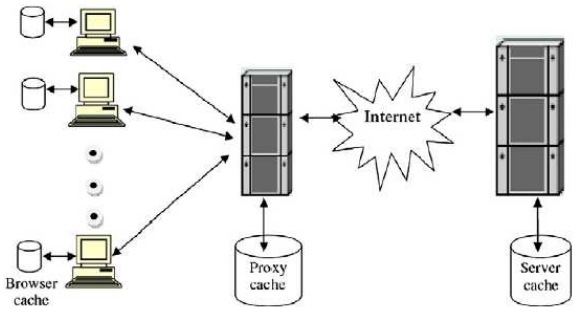

Figure 1: Different types of caches

Traditional Web proxy caching algorithms

Cache algorithms (also frequently called replacement algorithms or replacement policies) are optimizing instructions – or algorithms – that a computer program or a hardware-maintained structure can follow, in order to manage a cache of information stored on the computer. When the cache is full, the algorithm must choose which items to discard to make room for the new ones. The most widely used replacement algorithms include Least Recently Used (LRU), Least Frequently Used (LFU), Most Recently Used (MRU), SIZE [9], Greedy-Dual-Size(GDS), Hybrid [11], Lowest Relative Value(LRV) [10] etc.

Least Recently Used (LRU) discards the least recently used items first. This algorithm requires keeping track of what was used when, which is expensive if one wants to make sure the algorithm always discards the least recently used item. General implementations of this

ISSN: 2277-9655
Impact Factor: 1.852

technique require keeping "age bits" for cache-lines and track the "Least Recently Used" cache-line based on age-bits. In such an implementation, every time a cache-line is used, the age of all other cache-lines changes. Least Frequently Used (LFU) counts how often an item is needed. Those that are used least often are discarded first. Most Recently Used (MRU) discards, in contrast to LRU, the most recently used items first. MRU algorithms are most useful in situations where the older an item is; the more likely it is to be accessed. SIZE policy [9] is one of the common Web caching policies that replace the largest object(s) from a cache when space is needed for a new object. Thus, a cache can be polluted with small objects which will not be accessed again. Williams et al. [9] presented taxonomy of cache retrieval policies, by means of trace-driven simulations they measure the maximum feasible hit ratio and byte hit ratio. They suggested that it would be much better to replace documents based on the size as this maximizes the hit ratio in each of their workloads. Cao and Irani [12] introduced Greedy-Dual-Size (GDS) cache replacement algorithm. This algorithm integrates locality along with cost and size factors. Greedy-Dual-Size tags a cost with every object and expels the object that has the lowest cost or size. Wooster and Abrams[11] proposed Hybrid cache replacement algorithm make use of a combination of multiple requirements such as maintaining in the cache documents from servers that take significant time to connect to, those that need to be fetched from the slowest links, those that have been accessed very frequently, and those that are small. Wooster and Abrams checked the performance of Hybrid algorithm alongside LRU, LFU and SIZE. Hybrid algorithm performed well when compared with traditional LRU, LFU and SIZE replacement algorithms. The Lowest Relative Value (LRV) cache replacement algorithm proposed by Rizzo and Vicisano [10] expels the object that has the lowest utility value. In LRV, the utility of a document is calculated adaptively on the basis of data readily available to a proxy server. Rizzo and Vicisano show that LRV performs better than LRU and can substantially better the performance of a cache that is of modest size.

Improved Web proxy caching algorithms

A.P. Foong, H. Yu-Hen, D.M. Heisey [15] proposed a logistic regression model (LR) to predict the future request. Then, the objects with the lowest re-access probability value were replaced first regardless of cost and size of the predicted object. T. Koskela, J. Heikkonen, K. Kaski [14] used Multilayer perceptron network (MLP) classifier in Web caching to predict the class of Web objects depending on syntactic features from HTML structure of the document and the HTTP responses of the server as inputs of MLP. The class value was integrated with LRU, known to be LRU-C, to optimize the Web cache. However, frequency factor was ignored the frequency factor in Web cache replacement decision. An integrated solution of back-propagation neural network (BPNN) as caching decision policy and LRU technique as replacement policy for script data object has been proposed by Farhan [13]. Recently W. Ali, S.M. Shamsuddin, A.S. Ismail [5] proposed three algorithms namely SVM–LRU, SVM–GDSF and C4.5–GDS which make use of the capability of Support vector machine (SVM) and decision tree (C4.5) to learn from Web proxy logs files and predict the classes of objects to be re-visited or not. The trained SVM and C4.5 classifiers were incorporated with traditional Web proxy caching algorithms to present new Web proxy caching approaches. They proved that their proposed SVM–LRU, SVM–GDSF and C4.5–GDS significantly improved the performances of LRU, GDSF and GDS respectively in terms of hit ratio and byte hit ratio.

Machine learning Techniques

Machine learning, a branch of artificial intelligence, concerns the construction and study of systems that can learn from data. For example, a machine learning system could be trained on email messages to learn to distinguish between spam and non-spam messages. After learning, it can then be used to classify new email messages into spam and non-spam folders. The core of machine learning deals with representation and generalization. The two areas Machine learning and data mining overlap in many ways: data mining uses many machine learning methods, but often with a slightly different goal in mind. On the other hand, machine learning also employs data mining methods as "unsupervised learning" or as a preprocessing step to improve learner accuracy.

Naïve Bayes

A naïve Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong (naïve) independence assumptions. In simple terms, a naïve Bayes classifier assumes that the presence or absence of a particular feature is unrelated to the presence or absence of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 3" in diameter. A naïve Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of the presence or absence of the other features. For some types of probability models, naïve Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naïve Bayes models uses the method of maximum likelihood; in other words, one can work with the naïve Bayes model without

ISSN: 2277-9655
Impact Factor: 1.852

accepting Bayesian probability or using any Bayesian methods.

Mathematically, Bayes theorem gives the relationship between the probabilities of A and B, P(A) and P(B), and the conditional probabilities of A given B and B given A, P(A|B) and P(B|A). In its most common form, it is:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}.$$

In the Bayesian (or epistemological) interpretation, probability measures a degree of belief. Bayes theorem then links the degree of belief in a proposition before and after accounting for evidence. For example, suppose somebody proposes that a biased coin is twice as likely to land heads as tails. Degree of belief in this might initially be 50%. The coin is then flipped a number of times to collect evidence. Belief may rise to 70% if the evidence supports the proposition.

For proposition A and evidence B,

(A), the prior, is the initial degree of belief in A.

(A|B), the posterior, is the degree of belief having accounted for B.

he quotient P (B|A)/P (B) represents the support B provides for A.

An advantage of Naïve Bayes is that it only requires a small amount of training data to estimate the parameters (means and variances of the variables) necessary for classification. Bayesian classifiers are based on Bayes theorem.

Decision tree

Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. It is one of the predictive modeling approaches used in statistics, data mining and machine learning. In data mining, a decision tree describes data but not decisions; rather the resulting classification tree can be an input for decision making. Decision tree learning is a method commonly used in data mining [16]. The goal is to create a model that predicts the value of a target variable based on several input variables.

A decision tree is a simple representation for classifying examples. Decision tree learning is one of the most successful techniques for supervised classification learning. Decision trees used in data mining are of two main types:

1. lassification tree analysis is when the predicted outcome is the class to which the data belongs.
2. egression tree analysis is when the predicted outcome can be considered a real number (e.g. the price of a house, or a patient's length of stay in a hospital).

Decision tree learning is the construction of a decision tree from class-labeled training tuples. A decision tree is a flow-chart-like structure, where each internal (non-leaf) node denotes a test on an attribute, each branch represents the outcome of a test, and each leaf (or terminal) node holds a class label. The topmost node in a tree is the root node. An example Decision tree is shown in Figure 2.

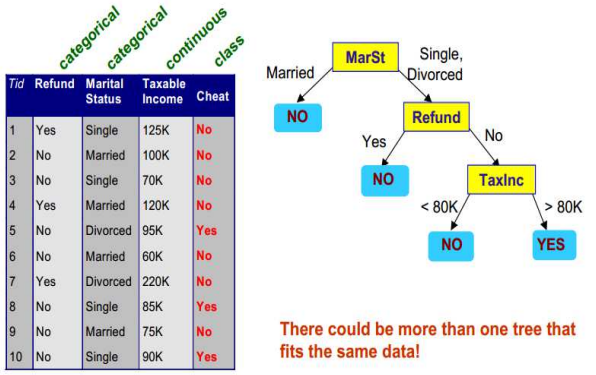

Figure 2: An example Decision tree

Amongst other data mining methods, decision trees have various advantages:

- imple to understand and interpret. People are able to understand decision tree models after a brief explanation.
- equires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed.
- ble to handle both numerical and categorical data. Other techniques are usually specialized in analyzing datasets that have only one type of variable.

The Proposed Web proxy Caching Algorithms

The proposed two algorithms namely NB–SIZE and C4.5–Hybrid make use of the capability of Naïve Bayes and Decision tree (C4.5) classifiers to learn from Web proxy logs files and predict the classes of objects to be re-visited or not. The trained Naïve Bayes and Decision tree (C4.5) classifiers were incorporated with traditional Web proxy caching algorithms to present new Web proxy caching approaches. Training and testing was done on Web proxy logs files (datasets) offline which can be used to predict the classes of objects to be re-visited or not in future. In order to prepare the training dataset, the desired features of Web objects are extracted from trace and logs proxy files. The important features of Web objects that indicate the user interest are extracted for preparing the training dataset. These features consist of URL ID, timestamp, elapsed time, size and type of Web object. The common features were selected and

ISSN: 2277-9655
Impact Factor: 1.852

extracted as suggested by W. Ali et al [5]. Subsequently, these features are converted to the input/output dataset or training patterns in the format <x1, x2, x3, x4, x5, y>. x1, …, x5 represent the inputs and y represents target output of the requested object. Table 1 shows the inputs and their meanings for each training pattern.

Table 1 : The inputs and their meanings

Input	Meaning
x1	Recency of Web object based
x2	Frequency of Web object
x3	Retrieval time of Web object
x4	Size of Web object
x5	Type of Web object

NB–SIZE

Williams et al. [9] suggested that it would be much better to replace documents based on the size as this maximizes the hit ratio in each of their workloads. As SIZE policy is replace the largest object(s) from a cache when space is needed for a new object. Thus, a cache can be polluted with small objects which will not be accessed again. Therefore, the Naïve bayes classifier is integrated with SIZE for improving the performance in terms of the hit ratio of SIZE. The proposed proxy caching policy is called NB–SIZE. In NB–SIZE, a trained Naïve Bayes classifier is used to predict the classes of Web objects either objects may be re-visited later or not. After this, the classification decision is integrated into cache replacement policy (SIZE) to give a value (unchanged/decreased) for each object in the cache. Consequently, the objects with the maximum size are removed first, there by postponing the removal of an object based on common factors of web object.

```
Begin
   For each web object Obj requested by user
   Begin
      if Obj in Cache
         Begin
            Cache hit occurs
            Update information of Obj
            //classify Obj by Naive Bayes classifier
            class of Obj=apply_NB(common features)
               If class of Obj=1 // Obj classified as revisited object in future
                  Decrease the size of object by the probability of frequency of object
               Else
                  Keep the original Size of object unchanged
         End

      Else
         Begin
            Cache miss occurs
            Fetch Obj from origin server.
            While no enough space in cache for Obj
            Begin
               Evict q such that q object is the biggest
            End
            //classify Obj by Naive Bayes classifier
            class of Obj=apply_NB(common features)
               If class of Obj=1 // Obj classified as revisited object in future
                  Decrease the size of object by the probability of frequency of object
               Else
                  Keep the original Size of object unchanged
         End
End
```

Figure 3: The proposed NB-SIZE algorithm

C4.5–Hybrid

Wooster and Abrams[11] proposed Hybrid cache replacement algorithm make use of a combination of multiple requirements such as maintaining in the cache documents from servers that take significant time to connect to, those that need to be fetched from the slowest links, those that have been accessed very frequently, and those that are small. Wooster and Abrams checked the performance of Hybrid algorithm alongside LRU, LFU and SIZE. Hybrid algorithm performed well when compared with traditional LRU, LFU and SIZE replacement algorithms. Though, Hybrid algorithm takes into consideration multiple factors of a web object, the capability of Decision Tree (C4.5) will boost the performance of the Hybrid algorithm. Therefore, the Decision Tree (C4.5) classifier is integrated with Hybrid algorithm for improving the performance in terms of the hit ratio of Hybrid algorithm. The proposed proxy caching policy is called C4.5–Hybrid. In C4.5–Hybrid, a trained Decision Tree (C4.5) classifier is used to predict the classes of Web objects either objects may be re-visited later or not. After this, the classification decision is integrated into cache replacement policy (Hybrid) to give a value (unchanged/decreased) for each object in the cache. Consequently, the objects with the maximum size and less frequently visited were removed first, there by postponing/proponing the removal of an object based on common factors of web object.

Implementation and Experimental Results

Raw data collection

The proxy logs files and traces of the Web objects were collected from [6], the two traces contain two month's worth of all HTTP requests to the NASA Kennedy Space Center WWW server in Florida. Two proxy datasets were collected between 2 from 00:00:00 July 1, 1995 through 23:59:59 July 31, 1995, a total of 31 days. The first log was collected from 00:00:00 July 1, 1995 through 23:59:59 July 31, 1995, a total of 31 days. The second log was collected from 00:00:00 August 1, 1995 through 23:59:59 August 31, 1995, a total of 7 days. In this two week period there were 3,461,612 requests. Timestamps have 1 second resolution.

Data pre-processing

Data pre-processing is an important step in the data mining process. The phrase "garbage in, garbage out" is particularly applicable to data mining and machine learning projects. Analyzing data that has not been carefully screened for such problems can produce misleading results. Thus, the representation and quality of data is first and foremost before running an analysis. Data pre-processing includes cleaning, normalization, transformation, feature extraction and selection, etc. The product of data pre-processing is the final training set.

ISSN: 2277-9655
Impact Factor: 1.852

```
Begin
    For each web object Obj requested by user
    Begin
        if Obj in cache
            Begin
                Cache hit occurs
                Update information of Obj
                //classify Obj by Decision Tree(C4.5) classifier
                class of Obj=apply_C4.5(common features)
                    If class of Obj=1 // Obj classified as revisited object in future
                        Decrease the size of object by the probability of frequency of object
                        Decrease the latency of object by the probability of frequency of object
                    Else
                        Keep the original Size of object unchanged
            End

        Else
            Begin
                Cache miss occurs
                Fetch Obj from origin server.
                While no enough space in cache for Obj
                Begin
                    Evict q such that q object is the biggest
                End
                //classify Obj by Decision Tree(C4.5) classifier
                class of Obj=apply_C4.5(common features)
                    If class of Obj=1 // Obj classified as revisited object in future
                        Decrease the size of object by the probability of frequency of object
                        Decrease the latency of object by the probability of frequency of object
                    Else
                        Keep the original Size of object unchanged
            End
End
```

Figure 4: The proposed C4.5-Hybrid algorithm

Training phase

Proxy log traces were preprocessed and training data sets were prepared based on the format requirement of the simulators. Each proxy dataset is divided randomly into training data (70%) and testing data (30%). Machine learning techniques are implemented using WEKA. Similarly Decision Tree (C4.5) is trained, J48 learning algorithm has been used, which is a Java re-implementation of C4.5 and provided with WEKA tool. The default values of parameters and settings are used as determined in WEKA. After training and verification, the trained classifiers were saved in the files which were utilized in improving the performance of the traditional Web proxy caching policies.

Web proxy cache simulation

The simulator software for non-uniform size web document caches was provided by University of Wisconsin [17]. The simulator can simulate LRU, SIZE, LRV, Hybrid and variations of Greedy Dual algorithms. The trained classifiers are integrated with simulator software to simulate the proposed Web proxy caching policies. The simulator takes input a text file describing each HTTP requests, calculates the hit ratio and byte hit ratio under an infinite-sized cache, and then calculates the hit ratio and byte hit ratio for each algorithm, under cache sizes being various percentages of the total data set size.

Performance Evaluation

Classifier evaluation

Three different performance measurements were used for evaluating the model/classifier. Table 4 shows each measure name and formula to calculate the same. Table 3 shows the values of performance measures of testing datasets. A correct classification rate (CCR) is a measure for evaluating a model or classifier. The true positive rate (TPR) or sensitivity, the true negative rate (TNR) or specificity also used to evaluate the performance of machine learning techniques. Table 2 shows the Confusion matrix.

Table 2: Confusion Matrix

	Predicted positive	**Predicted negative**
Actual positive	True positive (TP)	False negative (FN)
Actual negative	False positive (FP)	True negative (TN)

Table 3: The performance measures of testing dataset.

Classifier	**Measure Name**	**Value**
Naïve Bayes	CRR	80.192
	TPR	77.731
	TNR	90.211
Decision Tree(C4.5)	CRR	80.286
	TPR	78.361
	TNR	91.762

Table 4: The measures used for evaluating performance of machine learning techniques.

Measure name	**Formula**
Correct classification rate	$\textbf{CCR} = \frac{TP+TN}{TP+FP+FN+TN}(\%)$
True positive rate	$\textbf{TPR} = \frac{TP}{TP+FN}(\%)$
True negative rate	$\textbf{TNR} = \frac{TP}{TN+FP}(\%)$

Evaluation of proposed Web proxy caching approaches

Performance measures

In Web proxy caching, hit ratio (HR) and byte hit ratio (BHR) are two widely used metrics for evaluating the performance of Web proxy caching policies [14]. HR is defined as the ratio of the number of requests served from the proxy cache and the total number of requests. BHR refers to the number of bytes served from the cache, divided by the total number of

ISSN: 2277-9655
Impact Factor: 1.852

bytes served. Usually HR and BHR work in opposite ways. The following table shows the Pure hit-rate, Byte hit-rate and Reduced latency for existing and proposed algorithms. Table 5 shows the Pure hit-rate, Byte hit-rate and Reduced latency for existing and proposed algorithms. Graph of the interpreted data can be seen in figures 5, 6, 7.

Table 5: Pure hit-rate, Byte hit-rate and Reduced latency for existing and proposed algorithms.

Algorithm	Cache Size (%)	Pure Hit-Rate	Byte Hit-Rate	Reduced Latency
SIZE	0.05%	0.033147	0.002002	0.015402
	0.50%	0.115869	0.02222	0.065325
	5.00%	0.346999	0.163001	0.205181
	10.00%	0.490958	0.291574	0.335651
	20.00%	0.649384	0.471895	0.492107
NB-SIZE	0.05%	0.062107	0.006787	0.038447
	0.50%	0.191171	0.058369	0.10096
	5.00%	0.581985	0.392189	0.414467
	10.00%	0.782487	0.649755	0.650486
	20.00%	0.9454	0.902054	0.890965
Hybrid	0.05%	0.03378	0.01246	0.10020
	0.50%	0.155405	0.144142	0.714796
	5.00%	0.38999	0.320199	0.843834
	10.00%	0.5098	0.434811	0.883889
	20.00%	0.674279	0.606778	0.933231
C4.5-Hybrid	0.05%	0.09166	0.083021	0.453576
	0.50%	0.289056	0.247136	0.804798
	5.00%	0.707428	0.652751	0.950628
	10.00%	0.86594	0.840018	0.985528
	20.00%	0.965385	0.960595	0.997593

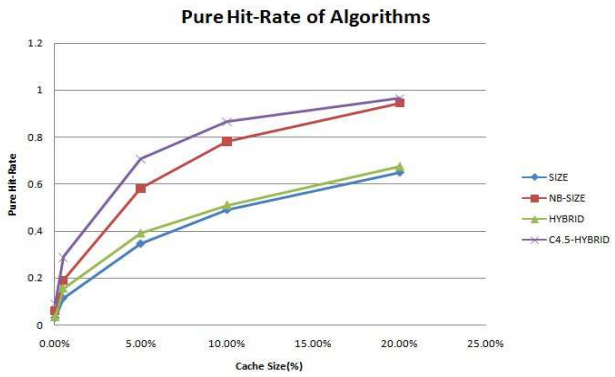

Figure 5: Pure hit-rate of existing and proposed algorithms.

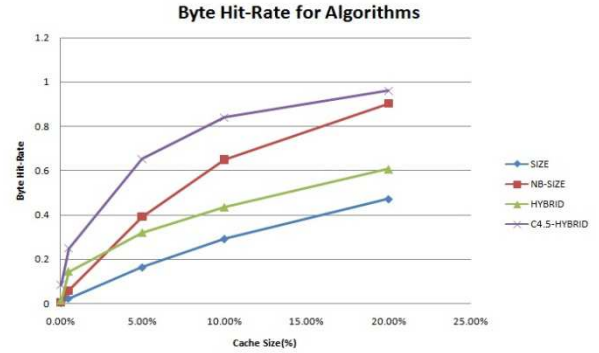

Figure 6: Byte hit-rate of existing and proposed algorithms.

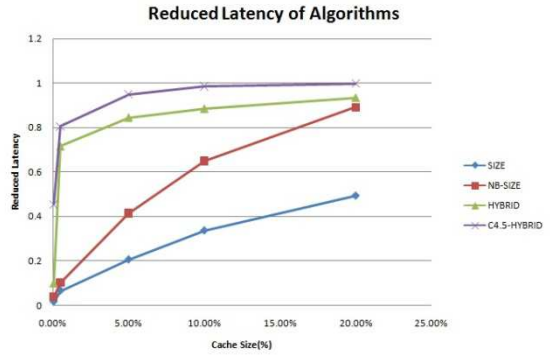

Figure 7: Reduced Latency of existing and proposed algorithms.

Conclusion

Experimental results have revealed that the proposed two algorithms NB–SIZE and C4.5–Hybrid significantly increased Pure Hit-Ratio, Byte Hit-Ratio and reduced the latency when compared with SIZE and Hybrid. The same is evident from the above graphs.

References

[1] H.T. Chen, Pre-Fetching and Re-Fetching in Web Caching Systems: Algorithms and Simulation, Trent University, Peterborough, Ontario, Canada, Peterborough, Ontario, Canada, 2008.

ISSN: 2277-9655
Impact Factor: 1.852

[2] *S. Romano, H. ElAarag, A neural network proxy cache replacement strategy and its implementation in the Squid proxy server, Neural Computing and Applications 20 (2011) 59–78.*
[3] *W. Kin-Yeung, Web cache replacement policies: a pragmatic approach, IEEE Network 20 (2006) 28–34.*
[4] *C. Kumar, J.B. Norris, A new approach for a proxy-level web caching mechanism, Decision Support Systems 46 (2008) 52–60.*
[5] *W. Ali, S.M. Shamsuddin, A.S. Ismail, Intelligent Web proxy caching approaches based on machine learning techniques, Elsevier (2012) 0167-9236.*
[6] *The University of California and Lawrence Berkeley National Laboratory, Traces: Available at http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html*
[7] *C. Van Koten, A.R. Gray, An application of Bayesian network for predicting object-oriented software maintainability, Inf. Softw. Technol. 48 (2006) 59–67.*
[8] *L. Rokach, O.Z.Maimon, Data Mining with Decision Trees: Theory and Applications, World Scientific, Singapore; Hackensack, NJ, 2008.*
[9] *M. Abrams, C.R. Standridge, G. Abdulla, E.A. Fox, S. Williams, Removal Policies in Network Caches for World-Wide Web Documents, ACM, 1996, pp. 293–305.*
[10] *Rizzo, Vicisano, Replacement Policies for a Proxy Cache, IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 8, NO. 2, APRIL 2000*
[11] *Roland P. Wooster and Marc Abrams: Proxy Caching That Estimates Page Load Delays from: Computer Networks and Isdn Systems - CN, Vol. 29, No. 8-13, pp. 977-986, 1997.*
[12] *Pei Cao and Sandy Irani, Cost-aware www proxy caching algorithms. In Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, California, December 1997.*
[13] *J. Cobb, H. ElAarag,Web proxy cache replacement scheme based on back-propagation neural network, Journal of Systems and Software 81 (2008) 1539–1558.*
[14] *T. Koskela, J. Heikkonen, K. Kaski, Web cache optimization with nonlinear model using object features, Computer Networks 43 (2003) 805–817.*
[15] *A.P. Foong, H. Yu-Hen, D.M. Heisey, Logistic regression in an adaptive Web cache, IEEE Internet Computing 3 (1999) 27–36.*
[16] *Rokach, Lior; Maimon, Data mining with decision trees: theory and applications. World Scientific Pub Co Inc. (2008) ISBN 978-9812771711.*
[17] *Copyright 1997. University of Wisconsin – Madison. All Rights Reserved.*