Designing Mathematical Model and Stochastic Hybrid Random Method for a Cache System to Create a Hybrid Cache Algorithm for Web Proxy to Increase the Speed of Accessing and Processing Data

Dr. Abdualmajed Ahmed Ghaleb Al-Khulaidi [1]

Dr. Mansoor Noman Marhoob Ali [2]

(1) (2) Ass. Professor, Internal medicine Computer science department

College of Education, Arts and Sciences - Marib

Sana'a University, Sana'a -Yemen


2003
١٤٢٤
جامعة الأندلس
للعلوم والتقنية
Alandalus University For Science & Technology

(AUST)

ISSN : 2410-7727

Designing Mathematical Model and Stochastic Hybrid Random Method for a Cache System to Create a Hybrid Cache Algorithm for Web Proxy to Increase the Speed of Accessing and Processing Data

Abstract :

This scientific article contains a mathematical model design and stochastic hybrid random method design for a cache system to get a hybrid cash algorithm to increase the speed of accessing, processing and saving data.. By this method , we can hybrid any kind of algorithms with each other. The simplicity of this method in hybridization is not only exclusive to web server but can be also used with databases, information systems, operating systems and CPU. Experiments were carried out to measure performance proficiency of hybrid cache algorithms by helping trace-driven simulation by comparing between our hybrid cache algorithm with famous hybrid cache algorithms(LRFU). Our hybrid algorithm contains three algorithms(GDS,LRU,LFU).Our hybrid algorithm was used in squid proxy that works with operation system Linux . This proxy server checks the data by the famous hybrid algorithm (LRFU) and our hybrid designed algorithm. The results of our hybrid designed algorithm indicate that Hit Ratio(HR) was 22% and Byte Hit Ratio (BHR) was 18% compared with famous hybrid algorithm(LRFU). This indicates that our hybrid cache designed algorithm is more efficient than famous algorithm(LRFU).

Keywords: Cache, Hit Ratio, Byte Hit Ratio ,Web Cache.

ISSN : 2410-7727

1. Introduction

When processing a big amount of data, a problem shows up, which is the slow access speed[1]. Caching is taking the data from the memory and saving it in the small cache which is much faster at delivering the data. In the contemporary time, raising performance and efficiency through cache algorithms has become an interested research topic, some scientists have committed some researches in this topic like.[2],[3],[4]: Sharieh A.,Castro M.,Dasarathan D., Chen Y.C., …etc. Hit ratio is a measurement for the cache's efficiency, it is referred to the availability of the necessary commands and data in the cache when the CPU directly searches for it, and when the CPU finds what it is looking for it is called a "hit", but when it does not find it it's called a "miss". In most modern devices the hit ratio nears 90%, this shows that the cache makes it faster. The access time to the memory in the computer can be improved which leads to a higher hit ratio, this means that the CPU will be retrieving what it needs from the cache directly rather than accessing the main memory, and this suggests that we need better cache algorithms[8],[9],[10]. In computing, a cache is a hardware or software component that stores data that can be served faster; the data stored in a cache might be the result of an earlier computation, or the duplicate of data stored elsewhere. A cache hit occurs when the requested data can be found in a cache, while a cache miss occurs when it cannot. Cache hits are served by reading data from the cache, which is faster than recomputing a result or reading from a slower data store; thus, the more requests can be served from the cache, the faster the system performs.

1.1.Cache Memory

Cache memory can be called CPU memory. This random access memory (RAM) that a computer microprocessor can access more quickly than regular RAM[11],[13],[17],[18]. This memory is typically integrated directly with the CPU chip or placed on a separate chip that has a separate bus interconnect with the CPU. The cache contains copies of the RAM, when the CPU reads a word from the RAM it first starts to look for it in the cache or the cache memory and if it can be

ISSN : 2410-7727

found it can be processed, but if it does not find it then it reads a block or a batch of words which gets written in the cache then the processing starts. The basic purpose of cache memory is to store program instructions that are frequently re-referenced by software during operations.

1.2.Web Proxy

Proxy is referred to a web agent, it consists of many devices and software that works in the network field, it works as an agent or as a middle man between the user and the other internet web servers and services and one of it's jobs is temporary caching of repeated requests from the user in the present time[1],[2]. The proxy is considered as special type of servers where it receives and processes the users' requests but it does not contain independent information rather bits and parts of the real servers' contents and acts as an agent for the real servers to display their contents.

1.2.1.Proxy Server Duties

As stated before, there are many types of proxies and they work in either of the following ways[20]:

1. Saving the most repeated requests and updating the currently saved contents, then delivering the cached contents to other users thus increasing the speed of replies and reducing traffic in the network.
2. Works as a firewall and traffic filter and manager the network since all the incoming or outgoing data of the network go through it.

The disadvantage of proxies is that sometimes the users who use the proxy will receive outdated data since the proxy replies to their requests using previously stored contents when the real contents have actually been dynamically altered in the real server without the proxy's knowledge; Some services in the network requests special proxies to reach the required real data which forces the local network to utilize other proxies in other networks to reach the desired data. Caching is a collection of frequently requested data by the users and it is stored locally, thus allowing a faster access to them by the users; it is seen as a small group of data for huge amount of information stored in the storage servers. A cache is always physically close from

ISSN : 2410-7727

the network internet user, today, cache systems are available in many modern personal computers and servers, many modern web browsers store frequently visited web pages by the user in the memory or other storage devices which allows other users to load those pages faster later on when he visits them. Web caching is a collection of locally saved web pages including the images, videos, texts and other types of documents that can be accessed using the HTT Protocol [15]. The web browser makes sure to prolong, maintain and update the web cache. The web cache works as a program installed in the local private servers that archives and recalling the frequently requested data by the user. If the requested documents are found by the proxy in the local cache then it retrieves it and registers a 'hit', however if the requested documents were not found then proxy registers a 'miss' and requests the documents from the real servers and transports it to the requested user[5]. Through the action of retrieving the data from a local server we get a faster response and reduce the time in the network and also increase the view range for every user. The proxy server allows the control of the contents being transferred so that bad pages would not be sent to the requester, it also frees space for new documents by deleting old and unused documents, this process occurs dynamically through using the modern LRU 'least recently used' algorithm. Web caching is the temporary storage of remote web objects on a local server. Advantages of this technique range from reduced access latencies to reduced server load and bandwidth consumption[16].

2. Algorithms Existing Cache

In computing, cache algorithms (also frequently called cache replacement algorithms or cache replacement policies) are optimizing algorithms — that a computer program or a hardware-maintained structure can follow in order to manage a cache of information stored on computer. When the cache is full, the algorithm must choose which items to be discarded to make a room for the new ones[16],[19],20].

2.1.Belady's Algorithm

The most efficient caching algorithm would be to discard the information that will not be needed for the longest time in the future. This optimal result is referred to as Belady's optimal algorithm or the

ISSN : 2410-7727

clairvoyant algorithm. Since it is generally impossible to predict how far in the future information will be needed, generally this is not implementable in practice.

2.2.Least Recently Used (LRU)

This cache algorithm keeps recently used items near the top of cache. Whenever a new item is accessed, the LRU places it at the top of the cache[4]. When the cache limit has been reached, items that have been accessed less recently will be removed starting from the bottom of the cache. This can be an expensive algorithm to use, as it needs to keep "age bits" that show exactly when the item was accessed. In addition, when a LRU cache algorithm deletes an item, the "age bit" makes changes on all the other items.

2.3.Most Recently Used (MRU)

This cache algorithm removes the most recently used items [6]. A MRU algorithm is good in situations in which the older an item is, the more likely it is to be accessed. Discards, in contrast to LRU, the most recently used items first.

2.4.Pseudo-LRU (PLRU)

For CPU caches with large associativity (generally >4 ways), the implementation cost of LRU becomes prohibitive[4][7]. In many CPU caches, a scheme that almost always discards one of the least recently used items is sufficient. So many CPU designers choose a PLRU algorithm which only needs one bit per cache item to work. PLRU typically has a slightly worse miss ratio, has a slightly better latency, and uses slightly less power than LRU.

2.5.Random Replacement (RR)

Randomly selects a candidate item and discards it to make space when necessary. This algorithm does not require keeping any information about the access history. For its simplicity, it has been used in ARM processors[7]. It admits efficient stochastic simulation[8].

ISSN : 2410-7727

2.6.Least-Frequently Used (LFU)

Counts how often an item is needed. Those that are used least often are discarded first. This cache algorithm uses a counter to keep track of how often an entry is accessed. With the LFU cache algorithm, the entry with the lowest count is removed first. This method isn't used that often, as it does not account for an item that had an initially high access rate and then was not accessed for a long time.

2.7.Adaptive Replacement Cache (ARC)

Constantly balances between LRU and LFU, to improve the combined result[10] [21]. ARC improves on SLRU by using information about recently-evicted cache items to dynamically adjust the size of the protected segment and the probationary segment to make the best use of the available cache space. Developed at the IBM Almaden Research Center, this cache algorithm keeps track of both LFU and LRU, as well as evicted cache entries to get the best use out of the available cache.

2.8.Clock with Adaptive Replacement (CAR)

Combines Adaptive Replacement Cache (ARC) and CLOCK. CAR has performance comparable to ARC, and substantially outperforms both LRU and CLOCK. Like ARC, CAR is self-tuning and requires no user-specified magic parameters[6].

2.9.Multi Queue (MQ)

The MQ cache contains multiple LRU queues, Q0, Q1, ..., Qm-1. Blocks stay in the LRU queues for a given lifetime, which is defined dynamically by the MQ algorithm to be the maximum temporal distance between two accesses to the same file or the number of cache blocks, whichever is larger [9] 12]. If a block has not been referenced within its lifetime, it is demoted from to Qi to Qi−1 or evicted from the cache if it is in Q0. Each queue also has a maximum access count; if a block in queue Qi is accessed more than 2i times, this block is promoted to Qi+1 until it is accessed more than 2i+1 times or its lifetime expires. Within a given queue, blocks are ranked by the recently of access, according to LRU.

ISSN : 2410-7727

2.10.Greedy Dual-Size (GDS) Algorithm

Greedy Dual-Size Algorithm incorporates locality with cost and size to decide which document to evict[14],[15]. This replacement policy can enhance hit ratio. Greedy Dual-Size algorithm takes into consideration factors such as locality, size and latency/cost. The algorithm is also a modified enhanced version of LRU. It is concerned with the case when pages in a cache have the same size, but incur different costs to fetch from a secondary storage. The algorithm associates a value, H, with each cached page p. Initially, when a page is brought into cache, H is set to be the cost of bringing the page into the cache (the cost is always non-negative). When a replacement needs to be made, the page with the lowest H value, minH, is replaced, and then all pages reduce their H values by minH . If a page is accessed, its H value is restored to the cost of bringing it into the cache. Thus, the H values of recently accessed pages retain a larger portion of the original cost than those of pages that have not been accessed for a long time. By reducing the H values as time goes on and restoring them upon access, the algorithm integrates the locality and cost concerns in a seamless fashion [17][18]. To incorporate the different sizes of the document, we extend the Greedy Dual algorithm by setting H to cost/size upon an access to a document, where cost is the cost of bringing the document, and size is the size of the document in bytes. We called the extended version the Greedy Dual-Size algorithm. The definition of cost depends on the goal of the replacement algorithm: cost is set to 1 if the goal is to maximize hit ratio, it is set to the downloading latency if the goal is to minimize average latency, and it is set to the network cost if the goal is to minimize the total cost. At the first glance, Greedy Dual-Size would require k subtractions when a replacement is made, where k is the number of documents in cache.

3. The Known Cache Algorithms Hybrid Method (LRFU)

To give an example of this, we have to look at the build of the hybrid algorithm that is made by fusing the LRU (Least recently used) algorithm and the Frequently used algorithm to make an LRFU algorithm[22]. In the LRFU algorithm for every object in the cache a

ISSN : 2410-7727

rating is calculated for the combined recency and frequency (CRF). To calculate the recursion, CRF recursion, the following function is used:

$$F(x) = \left(\frac{1}{p}\right)^{\lambda x} \qquad (1)$$

Where $p \geq 2$ is a variable which is equals near 2.

λ = control variable.

Where : x between 2 of the neighboring elements that were called. The rating is calculated for every element and the least rated element gets deleted separately from the cache . In it has been proven that when $\lambda = 0$ then all of the CRF results will be exactly the same as LFU.

The performance of the LRFU was studied based on the trace driven simulation example by designing hybrid methods for caching proxy servers as a temporary storage for web pages, this makes it the future requests get a faster reply. The temporary cache memory in the browser stores websites' data when visiting them, when the web resources is visited or required again the browser loads it from the saved cache which is much faster than connecting to the real server and downloading it again, but the resources in the cache can be old and obsolete and needs to be deleted so new data will get cached. To delete the temporary cached data in Firefox you need to do the following:

Open the menu by clicking on the button, then choose 'History'.

1. Click on the 'Clear Recent History'.
2. Choose the data you want to delete from the form, set it to the start to delete everything and tick the 'Cache' option.
3. To commit the action press the 'Clear Now' button.

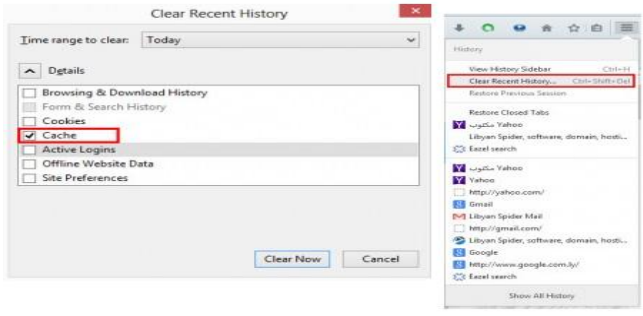


Figure.1. "Delete the temporary cached data in Firefox"

ISSN : 2410-7727

4. The Mathematical Model of Designing the Hybrid Cache Method for Web Proxy
5. Let's look at the mathematical model for a hybrid cache method under the issue of combining two cache algorithms.

We'll assume that an information system possesses objects that are represented as $N=\{1,2\ldots n\}$ (2)

The system contains a cache system with cache memory:

$$M=\{1,2\ldots m\} \quad (3)$$

The cache system can be found where there is a request for an object.

$$w=(r1,r2,\ldots,\ r_t,\ r_T),\ r_t \in N \quad (4)$$

Where Object-r_t belongs to a certain N which is a request from the cache in a certain time 't'

We symbolize the set of the subsets where the object N is present in the Cache memory(M) as:

$$\mathcal{M}_m$$

$$\mathcal{M}_m = \{S|S \subseteq N \wedge |S| \leq m\} \quad (5)$$

When making a decision on solving a problem in a cache system you can use two algorithms, we'll assume that the following cache system: Cache $A_1$and A_2 where they are:

$$A_1 = (Q_1, q_{10}, g_1), \quad (6)$$

$$A_2 = (Q_2, q_{20}, g_2), \quad (7)$$

Where Q_1 and Q_2 is a set of management algorithms for A_1 and A_2. Where q_{10} and q_{20} are the initial state for managing the algorithm for A_1 and $A_{2..}$ The process flow for A_1 and A_2 which based on the cache's state and the management algorithms state Q_1 and Q_2 would be the g_1 and g_2 algorithms. A new state for the cache and the algorithms for Q_1 and Q_2 also A_1 and A_2 of management

$$g_1: \mathcal{M}_m \times Q_1 \times N \to \mathcal{M}_m \times Q_1; \quad (8)$$

$$g_1(S,q_1,x) = \begin{cases} (S, q_{11}), & if \quad x \in S; \\ (S \cup \{x\}, q_{11}), & if \quad x \notin S \wedge |S| < m; \\ (S \cup \{x\}\{y\}, q_{11}), if\ (x \notin S) \wedge (|S| = m) \\ \quad \wedge (y \in S) \wedge \big(y = d_1(S, q_1)\big); \end{cases} \quad (9)$$

$$d_1: \mathcal{M}_m \times Q_1 \to N \quad (10)$$

This is in the state of A_1 algorithm

ISSN : 2410-7727

And for in the state of A_2 algorithm:

$$g_2: \mathcal{M}_m \times Q_2 \times N \to \mathcal{M}_m \times Q_2; \quad (11)$$

$$g_2(S, q_2, x) = \begin{cases} (S, q_{21}), & if \quad x \in S; \\ (S \cup \{x\}, q_{21}), & if \quad x \notin S \wedge |S| < m; \\ (S \cup \{x\}\{y\}, q_{21}), & if \ (x \notin S) \wedge (|S| = m) \\ \wedge (y \in S) \wedge \big(y = d_2(S, q_2)\big); & \end{cases} \quad (12)$$

$$d_2: \mathcal{M}_m \times Q_2 \to N \quad (13)$$

The hybrid algorithm A for the A_1,A_2 algorithm will be called the organized set

$$A = (Q_1, Q_2, q_{10}, q_{20}, g), \quad (14)$$

The process flow of the hybrid algorithm g

$$g: \mathcal{M}_m \times Q_1 \times Q_2 \times N \to \mathcal{M}_m \times Q_1 \times Q_2 \quad (15)$$

Then:

$$\big(S_1, q_{11}, q_{21,}\big) = g(S, q_1, q_2, x). \quad (16)$$

$$g_1(S, q_1, x) = \begin{cases} (S, q_{11}, q_{21}), & if \quad x \in S; \\ (S \cup \{x\}, q_{11}, q_{21}), & \\ if \quad x \notin S \wedge |S| < m; & \\ (S \cup \{x\}\{y\}, q_{11}, q_{21}), & \\ if \ (x \notin S) \wedge (|S| = m) \wedge & \\ (y \in S) \wedge \big(y = R(S, q_1, q_2)\big); & \end{cases} \quad (17)$$

Where a certain Object-X has a request in a time 't'.

S – The state of the cache memory at the time 't'.

q_1- The state of the managing algorithm A_1 at the time 't'.

q_2- The state of the managing algorithm A_2 at the time 't'.

q_{11}- The state of the managing algorithm A_1 at the time 't+1'.

q_{21}- The state of the managing algorithm A_2 at the time 't+1'. And in the event of a hybrid algorithm R it will be the following: $R: \mathcal{M}_m \times Q_1 \times Q_2 \to N$ (18)

In the state R of the cache memory and a state q_1,q_2 for the managing algorithm and the algorithms A_1,A_2 it is possible to choose an Object-Y to show state. The cache memory S. From this model we can see that the main problem of hybridization is summarized by choosing R which allows the selection of the management state q_1,q_2 for the algorithm A_1,A_2 . To manage the influential power of the algorithms

ISSN : 2410-7727

A_1,A_2 we provide the normal management concept. We'll assume a variable without an influential power that represents the algorithms A_1,A_2 for deciding the decision R. $\lambda \in [0,1]$. The variable will be the definition of the variable of managing the influential power for the algorithms A_1,A_2. In the state of [0,1] symbolized by **I**:

It will become **I**=[0,1] and R's syntax will be:

$$R: \mathcal{M}_m \times Q_1 \times Q_2 \times I \to N, y = R(S, q_1, q_2, \lambda) \qquad (19)$$

Then R will have the following attributes:

$$R(S, q_1, q_2, 0) = d_2(S, q_2) \qquad (20)$$

$$R(S, q_1, q_2, 1) = d_1(S, q_1) \qquad (21)$$

Where: d_1,d_2 will reflect :

$$d_1: \mathcal{M}_m \times Q_1 \to N \qquad (22)$$

$$d_2: \mathcal{M}_m \times Q_2 \to N \qquad (23)$$

In this state showing the process flow of the hybrid algorithm A for A_1,A_2 is:

$$g: \mathcal{M}_m \times Q_1 \times Q_2 \times I \times N \to \mathcal{M}_m \times Q_1 \times Q_2 \qquad (24)$$

Thus the general state will be:

$$(S_1, q_{11}, q_{21}) = g(S, q_1, q_2, \lambda, x) \qquad (25)$$

Where λ the influential power management variable – A_1,A_2 in 't'.

$$(S, q_1, q_2, \lambda, x) = \begin{cases} (S, q_{11}, q_{21}), & if \quad x \in S; \\ (S \cup \{x\}, q_{11}, q_{21}), & if \quad x \notin S \wedge |S| < m; \\ (S \cup \{x\}\{y\}, q_{11}, q_{21}), if\ (x \notin S) & \\ \wedge\ (|S| = m) \wedge (y \in S) \wedge \big(y = R(S, q_1, q_2, \lambda)\big); & \end{cases} \qquad (26)$$

$$R: \mathcal{M}_m \times Q_1 \times Q_2 \times I \to N \qquad (27)$$

To achieve the goal of the article we suggest acquiring the hybrid algorithms through random strategy method.

5.Algorithms Hybridization Method

Let's look at the idea using an example of hybridizing two cache algorithms. When solving the selection f the random object we will either use A_1 or A_2 algorithm. The control variable is the probability of choosing algorithm A_1 . We'll assume at time 't' the probability choosing the algorithm A_1 in solving a problem is: $\lambda, 0 \le \lambda \le 1$ .Then to solve this equation using the A_2 algorithm it must be chosen with a probability of 1 - λ, we compare the random value '4' with the

ISSN : 2410-7727

time [0,1] using the distribution law, if the value '4' in the time 't' is less than 'ʎ 'then we will use the algorithm A_1, this means that outcome of the random value: $d_1(S, q_1)$. If the value '4' in the time 't' is less than xx then the algorithm used to solve this will be A2, that means that the outcome of the random value : $d_2(S, q_2)$. We will call the suggested method for hybridization as 'Stochastic Hybrid Method'. The hybrid algorithm A was built on A_1,A_2 's bases.

$$A = (Q_1, Q_2, q_{10}, q_{20}, \lambda_0, g) \qquad (28)$$

The initial value for the control variable in the hybrid algorithm A $.\lambda_0 \in I = [0,1]$. The process flow of the algorithm g:

$$g: \mathcal{M}_m \times Q_1 \times Q_2 \times I \times N \to \mathcal{M}_m \times Q_1 \times Q_2 \qquad (29)$$

$$g_1(S, q_1, q_2, \lambda, x) = \begin{cases} (S_1, q_{11}) = g_1(S_1, q_1, x);\ (S_1, q_{21}) \\ \quad = g_2(S_1, q_2, x); \\ \quad if\ 0 \le \zeta < \lambda; \\ (S_1, q_{21}) = g_2(S_1, q_2, x); \\ \quad (S_1, q_{11}) \\ \quad = g_1(S_1, q_1, x); \\ \quad if\ \lambda \le \zeta \le 1; \end{cases} \qquad (30)$$

Where : $S_1 -$ cache memory state at time 't+1'.

$\mathbf{X} = \mathbf{r_t}$ – Object request in time 't'.

Where:

$$x \in S_1$$
$$\zeta \in I$$

Random number which it's probability equals to the distribution law at the time I=[0,1].

6.Hybrid Cache Algorithm for Proxy Server

If we look at the foundations of the LFU and LRU, LRU was designed based on LRU-Queue, in this situation the Hit Ratio is calculated using the following algorithm:

$$CV_{LRU} = \frac{1}{(T - T_1)} \qquad (31)$$

Where:

T is the real time.

$\mathbf{T_i}$ is the last time of arrival of the object.

ISSN : 2410-7727

The LFU algorithm was built based on the count of requests of the object and is calculate with:

$$CV^{i}{}_{LFU} = N_i \qquad (32)$$

The GDS was specially developed for web systems and servers, the Hit Ratio is calculated with:

$$CV^{i}{}_{GDS} = \frac{C_i}{S_i} \qquad (33)$$

Where:

C_i is the access value for the object

S_i is the size of the object.

Hybrid algorithm processing flow:

1. The object with the lowest value of the Hit Ratio will be replace.
2. The value of the Hit Ratio for each object in the cache memory decrease depending on the object with the lowest value of the Hit Ratio.
3. Repeat the first and second step until the cache has been freed ,then it allows to save temporary the new object.
4. The new object is cached and the hit ratio is calculated using the previous formula:

$$CV^{i}{}_{Hybrid} = \frac{P_i . C_i}{S_i} \qquad (34)$$

Where:

C_i is the access value for the object .

S_i is the size of the object.

P_i is the requests of the object depend on **:**

T , T_i , N_i .

Where:

T is the real time .

T_i is the last time of arrival of the object.

N_i count of requests of the object **.**

Let N be the total number of requests (objects).

ISSN : 2410-7727

δi= 1, if the request i is in the cache, while δi= 0, otherwise Mathematically, this can be expressed as follows:

$$HR = \frac{\sum_{i=1}^{N} \delta_i}{N} \quad (35)$$

$$BHR = \frac{\sum_{i=1}^{N} b_i \, \delta_i}{\sum_{i=1}^{N} b_i} \quad (36)$$, where bi is the size in bytes of the ith requested object.

7. Using the Mathematical Model and Stochastic Hybrid Random Method for the Trace-Driven Simulation by Using Hybrid Cache Algorithm

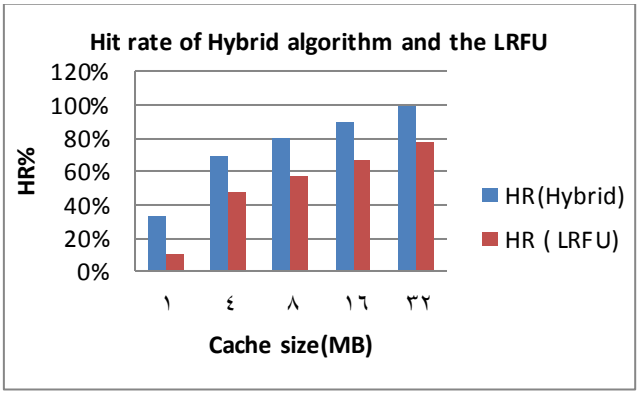
We did the trace-driven simulation to compare the performance of our proposed hybrid cache algorithm with LRFU cache replacement algorithm. Hybrid cache algorithm has used in Squid proxy which is open source and work in operating system environment Linux to improve the temporary storing and replace the web pages in proxy. Squid proxy is open source web proxy and program code and it is available for development and work as a medium between web browsers, for users and servers containing data. Also it manages servers and users and produces caching. The proxy server keeps the repeated data in users request and storage at speeding the work and reducing to pass in the network. To fix the proxy on Google chrome, we follow:

1. Enter Google chrome settings.
2. Press show advanced settings.
3. Press change proxy settings.
4. Internet setting page will appear ,then press connection then Lan setting.
5. Tick true in the square proxy server then press advanced.
6. Write proxy server then tick true on the square use the same proxy server then press ok.

ISSN : 2410-7727

"Table 1. Hit rate of Hybrid algorithm and the LRFU".

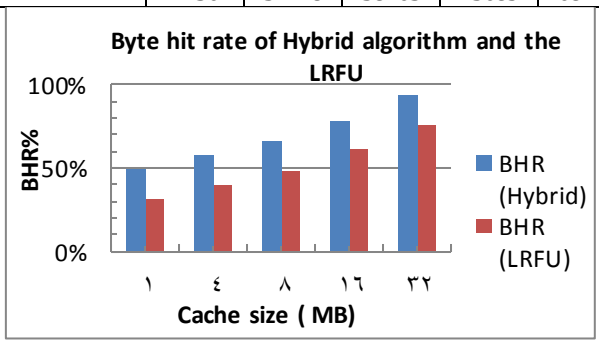
Total Cache size set is(in MB)	1	4	8	16	32
HR(Hybrid)	0.3299 69366	0.6904 66168	0.7971 41187	0.8981 63779	0.9993 08424
HR (LRFU)	0.1078 09655	0.4717 24698	0.5703 69017	0.6741 46635	0.7749 41651



"Figure.2. Hit ratio (HR) for the proxy trace"

"Table 2. Byte hit rate of Hybrid algorithm and the LRFU".

Total Cache size set is(in MB)	1	4	8	16	32
BHR(Hybrid)	0.498 6948	0.580 45623	0.666 51292	0.787 36366	0.938 80836
BHR (LRFU)	0.319 2456	0.402 34718	0.484 38765	0.614 13689	0.753 68176



"Figure.3. Byte hit ratio (BHR) for the proxy trace"

ISSN : 2410-7727

Hit Ratio (HR), Byte Hit Ratio (BHR) are the most widely used metrics in evaluating the performance of Web caching. HR is defined as the percentage of requests that can be satisfied by the cache. BHR is the number of bytes satisfied from the cache as a fraction of the total bytes requested by user. Total Cache size set is 1MB,4MB, 8MB,16MB,32MB. The results of our hybrid cache algorithm indicate that Hit Ratio(HR) was 22% and Byte Hit Ratio (BHR) was 18% compared with famous hybrid cache algorithm (LRFU). This indicates that our hybrid cache designed algorithm is more efficient than famous hybrid cache algorithm (LRFU).

8.Conclusion

In this scientific article, a mathematical model design and stochastic hybrid random method design for a cache system to get a hybrid cash algorithm to increase the speed of accessing, processing and saving data. The hybrid cache method can hybridize any type of cache algorithms with each other. The simplicity of this method in hybridization is not only exclusive to web server but can be also used with databases, information systems, operating systems and CPU. Experiments were carried out to measure performance proficiency of hybrid cache algorithms by helping trace-driven simulation by comparing between our hybrid cache algorithm with famous hybrid cache algorithm(LRFU). The results of our hybrid designed algorithm indicate that Hit Ratio(HR) was 22% and Byte Hit Ratio (BHR) was 18% compared with famous hybrid algorithm(LRFU). This indicates that our hybrid cache designed algorithm is more efficient than famous algorithm(LRFU).

ISSN : 2410-7727

9. References

[1].Sirshendu Sekhar Ghosh and Dr. Aruna Jain, "Web Latency Reduction Techniques: A Review Paper", IJITNA vol.1 No.2 pp20 – 26, September, 2011.

[2].Sirshendu Sekhar Ghosh, Dr. Aruna Jain, "Hybrid Cache Replacement Policyfor Proxy Server", International Journal of Advanced Research in Computer and Communication Engineering Vol. 2, Issue 3, March 2013 .

[3]. M. Dehghan, L. Massoulie, D. Towsley, D. Menasche, and Y. Tay, "A Utility Optimization Approach to Network Cache Design," in Proc. of IEEE INFOCOM 2016, 2016, to appear, arXiv preprint arXiv:1601.06838.

[4].C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for LRU cache performance," in Proceedings of the 24th International Teletraffic Congress, 2012,p. 8.

[5].N. C. Fofack, P. Nain, G.Neglia, and D.Towsley, Performance evaluation of hierarchical TTL-based cache networks,"Computer Networks,vol. 65, pp.212 – 231, 2014.

[5]. JyotiPandey, AmitGoel, Dr. A K Sharma "A Framework for Predictive Web Prefetching at the Proxy Level using Data Mining" IJCSNS,VOL.8 No.6, pp303-308, June 2008.

[6]. Shaul Dar, Michael J. Franklin, BjörnÞórJónsson, DiveshSrivastava, and Michael Tan. Semantic Data Caching and Replacement. VLDB, 1996.

[7]. A.Balamash, M.Krunz, "An Overview of Web Caching Replacement Algorithms",IEEE Communications Surveys & Tutorials, Second Quarter, 2004.

[8]. Nimrod Megiddo and Dharmendra S. Modha. ARC: "A Self-Tuning, Low Overhead Replacement Cache". FAST, 2003.

[9]. Yuanyuan Zhou, James Philbin, and Kai Li. The 23Multi-Queue Replacement Algorithm for Second Level Buffer Caches. USENIX, 2002.

[10]. Eduardo Pinheiro , Ricardo Bianchini, Energy conservation techniques for disk array-based servers, Proceedings of the 18th annual international conference on Supercomputing, June 26-July 01, 2004, Malo, France.

[11].Cheng Li, Philip Shilane, Fred Douglis and Grant Wallace. Pannier: A Container-based Flash Cache for Compound Objects. ACM/IFIP/USENIX Middleware, 2015.

ISSN : 2410-7727

[12]. P. Cao, E.W. Felten, A.R. Karlin, K. Li, "A Study of Integrated Prefetching and Caching Strategies", In Proc. Of the ACMSIGMETRICS Conference on Measurement and Modeling of Computer Systems, pp. 188-197, 1995.

[13] N. Beckmann and D. Sanchez, "Jigsaw: Scalable software-defined caches," in PACT-22, 2013.

[14] Q.Yang, J.Z.Huang, N.Michael, "A Data Cube Model for Prediction-Based Web Prefetching", Journal of Intelligent Information Systems, 2003,vol.20 (1),pp.11-30.

[15].W.-G. Teng, C.-Y. Chang, and M.-S. Chen, "Integrating Web caching and Web prefetching in client-side proxies," IEEE Transactions on Parallel and Distributed Systems, vol. 16, no. 5,pp. 444–455, 2005.

[16].C. Bouras, A. Konidaris, and D. Kostoulas, "Predictive prefetching on the Web and its potential impact in the wide area." World WideWeb, vol. 7, no. 2, pp. 143–179, 2004.

[17]. N. Beckmann and D. Sanchez, "A cache model for modern processsors," MIT, Tech. Rep. MIT-CSAIL-TR-2015-011, 2015.

[18] Bhawna Nigam and Dr. Suresh Jain, "Analysis of Markov Modelon different Web Prefetching and Caching schemes", IEEE, 2010.

[19].L. Shi, B. Song, X. Ding, Z. Gu, and L. Wei, "Web prefetching control model based on prefetch-cache interaction," in Proc. First International Conf. on Semantics, Knowledge and Grid SKG '05,2005.

[20]. Jyoti Pandey, Amit Goel, Dr. A K Sharma "A Framework forPredictive Web Prefetching at the Proxy Level using Data Mining"IJCSNS,VOL.8 No.6, 303-308, June 2008.

[21].F.J. González-Cañete, E. Casilari, A. Triviño-Cabrera "A contenttypebased evaluation of Web Cache replacement policies" IADISInternational Conference Applied Computing 2007.

[22].L .Donghee, H. N. Sam, C. Jongmoo, L. M. Sang, "Implementation and Performance Evaluation of the LRFU Replacement Policy", Dep. Of Computer Engineering, Seoul National University, Seoul 151-742 Korea.