Journal of Engineering and Applied Sciences 12 (Special Issue 10): 9013-9020, 2017
ISSN: 1816-949X
© Medwell Journals, 2017

Performance Improvement of Proxy Server Cache Replacement by Combination FIFO-LRU-LFU Algorithms

[1, 2]Tanwir, [1]Gamantyo Hendrantoro and [1]Achmad Affandi
[1]Department of Electrical Engineering,
Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia
[2]Department of Electrical Engineering,
Universitas Sains dan Teknologi Jayapura, Jayapura, Indonesia

Abstract: Cache herein refers to a storage for results of internet browsing located in cache disk. Size of cache and choice of cache replacement algorithm influence the system speed and client access throughput. Arbitrary deletion of objects during cache replacement can lead to loss of objects frequently used which result in a miss when there is a new request. In order to improve throughput an algorithm combining three cache replacement algorithms, FIFO-LRU-LFU is proposed. The new algorithm combine constructively advantages of the three different algorithms. Analysis has been conducted to understand the effect of cache size on hit rate percentage, response time, delay time and throughput when the FIFO-LRU-LFU algorithm is implemented. The results indicate improvement of bandwidth efficiency in cache replacement process compared to either single algorithm or double combination algorithms which reaches 87% of rate hit percentage. As a result, there is a decrease in bandwidth use and delay time which results in increased hit rate. In addition, there is an increase of relative throughput compared to double algorithm of approximately 95% when 100 MB cache is used and 83% with 200 MB cache. The increase of this throughput influences the efficiency of bandwidth use and reduces delay time to only 25.6% with 100 MB cache and 11.8% with 200 MB cache. This result is in favour of the adoption of the FIFO-LRU-LFU for cache replacement algorithm.

Key words: Cache, cache replacement algorithm, hit rate, internet networking, proxy server, throughput

INTRODUCTION

Information technology grows rapidly in line with the growth of computer networking technology, particularly internet. One of the important elements in internet access is proxy server, a server that can be configured for several functions which include cache server and bandwidth controller (Shim *et al*., 1999). Cache in this case is temporary storage for objects to accelerate object transfer in the proxy server. With cache, every object request is not directly served by accessing internet but initially by accessing buffer in the cache. When the object requested is found here the response time of the request will be faster and the use of channel bandwidth can be minimized (Meira *et al*., 1998). As the cache size is limited, the use of bandwidth is controlled by deleting cached object by using cache replacement algorithmsn (Zhang *et al*., 2004). The deleting process is done when the cache has been full and a new saving entry is needed.

The storing process of objects at proxy cache is done by saving objects have been accessed (Wang *et al*., 2004; Chang *et al*., 2008). When users request objects that have been accessed previously in the internet which has been stored in the cache, the proxy server will directly provide the objects from the cache without re-requesting from server source. However, when the requested objects are not found in the cache of proxy server, the proxy server will forward the request to the source server in the internet (Lim *et al*., 2014; Vladimir *et al*., 2017). The proxy server herein functions as cache storage for objects that are possibly requested later by client computer to public internet. Through HTTP mechanism, objects given by proxy are always the newest object as the proxy server always matches the existing objects in the cache with objects in the source server. Meanwhile, the validity and effectiveness of cache replacement algorithm is used to analyse and compare work performance of proxy server based on request hit, byte hit and response time (Lee and Kim, 2010). Three algorithms which are frequently used in cache replacement process are FIFO (First In First Out), LRU (Least Recently Used) dan LFU (Least Frequently Used). FIFO algorithm is the simplest one as it is similar to unprioritised queue. Earlier incoming page will also come out earlier. This algorithm uses stack data structure when

Corresponding Author: Tanwir, Department of Electrical Engineering, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia there is no empty page while page fault occurs, frame at the lowest stack is chosen. Therefore, it could happen that this algorithm moves pages frequently used. The weakness of FIFO is that it does not perform well all of the time. This is because there is a possibility that a page that has been removed from memory is requested again (Jacob *et al*., 2010; Lenjani and Hashemi, 2014).

LRU algorithm selects pages that have not been used for longest time. The advantage of LRU is that it does not experience Belady anomaly and only discharges objects that have never been accessed for a long time without considering when those objects came in the cache disk for the first time (Stallings, 2000; Akon *et al*., 2010). The implementation of LRU (Least Recently Used) is done by using stack indicat ing the objects in the cache disk. Every time an object is accessed, it will be positioned at the top of the cache disk. When an object at the lowest part of the stack has passed the maximum time limit this object will be replaced. Every time a new object is being accessed, the object to be replaced is determined based on its expiry time. For every appearance of a new object, the algorithm conducts loop twice; firstly, it looks for the oldest object and secondly, it replaces object, so that, it needs a longer time in replacing object compared to FIFO (First In First Out).

LFU algorithm replaces the least used pages. LFU (Least Frequently Used) detects unused objects in a certain period of time with fetch count = 4. As such LFU algorithm is simple and efficient, since, it does not require many steps in selecting objects (Liu *et al*., 2007; Aghaei and Zaman-Zadeh, 2016). The weakness, however is that the objects discarded due to least frequent use might possibly be in active use.

In the previous research, combination of FIFO and LRU algorithms has been proposed to exploit advantages of each algorithm. The combined algorithm is subsequently compared to the single algorithms in terms hit rate, response time and cache delay (Nakamura *et al*., 2002; Anandharaj and Anitha, 2009). It was found that using the combined algorithm bandwidth efficiency increases in the cache replacement process compared to single algorithm in which throughput in the server network improves with cache size. The combined algorithm also reduces the bandwidth use resulting in reduction of delay time, smaller miss rate and higher hit rate.

Literature review: Hendrantoro and Affandi (2015) indicate that the combination of three algorithms LRU, LFU and FIFO at cache servers in telecommunication networks promises improvement of work performance in cache replacement in the internet and heavily influences network bandwidth use and cache hit rate. From the earlier findings, further study has been carried out by combining three algorithms and evaluating the improvement of researcher performance which is reported in this study. Therefore, the original contribution of this research includes the combination of three algorithms LRU, LFU and FIFO and its performance evaluation which have never been reported elsewhere. The evaluation shows improvement in work performance of cache replacement at proxy server compared to both single and double combination algorithms (Tanwir *et al*., 2017). As a result, there is an improvement of throughput and bandwidth efficiency as well as reduction of miss rate.

MATERIALS AND METHODS

Cache proxy plays an important role in improving researcher performance of a computer network. Proxy is an aplication that serves as an intermediate between a client and web server (Ji *et al*., 2015; Ajorloo and Manzuri-Shalmani, 2016) as shown in Fig. 1. One of the functions of a proxy is to store cache.

In LAN networking when a client accesses a web URL and requests an object, the browser will send the request to proxy server. However, if that object is not available then the proxy server will directly forward the request to the web server in order to accelerate browsing process.

A computer system consists of CPU, RAM, HDD and others in which cache memory is a small size high speed memory in CPU used to store copies of objects or instructions frequently accessed by CPU. Our research, however, focuses on cache disk located at proxy server as illustrated in Fig. 2. The basic function of a cache disk at proxy server is to store objects that have been accessed through internet network. As such when a user requests an internet service containing objects which have previously been accessed or requested, i.e., have supposedly been in the cache then the proxy server will be able to provide them from the cache instantly to the user without re-requesting from internet server.

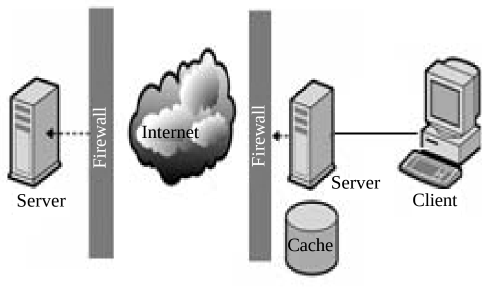

Fig. 1: Client-server relationship

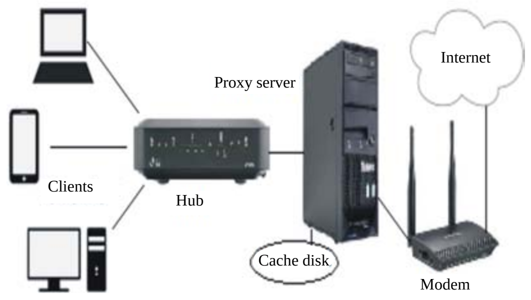

Fig. 2: Client-server relationship

Cache functions as temporary storage. When there is an object request, the system will firstly search for it in the cache. When the requested data is found the processor will read it with a small delay. However, if the data is not found, the processor will search for it in RAM which has lower speed. A cache disk can provide data requested by processor, so that, the impact of slow work of RAM can be mitigated. In this manner, bandwidth efficiency will increase and the processor will work more efficiently. Furthermore, the cache disk with bigger capacity will also increase overall computer researcher speed.

The research principle of a cache is basically to determine whether or not an object has been requested previously. In general, it is done by checking if the same query has been made before. If the query has been done, cache hit will occur and then the search result from the cache will be returned to user. However, if the query has never been made previously, cache miss will occur and then searching will proceed in the network.

Algorithms and evaluation methods: This study describes implementation of cache replacement algorithms in the form of pseudo-codes and performance evaluation method using an experimental network with 3 clients.

FIFO algorithm: FIFO algorithm performs replacement of objects that have existed for a long time in the cache. The oldest objects are assumed not to be accessed anymore and therefore will be replaced by the new ones. The weakness is that the algorithm will discard objects that have expired although they might be still actively used. FIFO applies a logic principle in which earlier incoming object will come out earlier too which is implemented in the pseudo-code in Algorithm 1 through the condition check “cache disk First In”.

Algorithm 1; Pseudo code of FIFO algorithm:

```
Read Xn; Yn; Zn
    If (cache disk First in) then,
            Print “cache disk
    If (New Object) then,
            Print “cache disk”
    Else
            Print “cache disk FIFO”
    End If
```

LRU algorithm: LRU algorithm researchers by removing objects that have never been used for a long time. The idea is the objects that have long been unused have small posibility to be requested again, so that, their place can be used by new objects which are more probably used in the future. LRU algorithm implements clock logic counter and stack. At counter, every object has an initial value of zero. When an object is accessed the clock for that object will increase and object with smallest clock value is replaced. With regards to the stack, on the other hand, every time an object is accessed this object will be positioned at the top of the stack. The lowest object of the stack will be replaced. In the pseudo-code of LRU as illustrated in Algorithm 2, the implementation of clock logic counter and stack is done in the checking stage of “cache disk Expires”.

Algorithm 2; Pseudo code of LRU algorithm:

```
Read Xn; Yn; Zn
    If (cache disk Expires) then,
            Print “cache disk
    If (New Object) then,
            Print “cache disk”
     Else
            Print “cache disk LRU”
    End If
```

LFU algorithm: LFU algorithm conducts replacement of objects in the cache which are the most seldom used. These objects are considered not to be used anymore and replaced with newer objects. The weakness of this algorithm is that these discarded objects might still be actively used. LRU applies a logical principle to replace objects which are most rarely used which is implemented in the pseudo-code in Algorithm 3 with condition check “cache disk Frequently” .

Algorithm 3; Pseudo code of LFU algorithm:

```
Read Xn; Yn; Zn
    If (cache disk Frequently) then,
            Print “cache disk
    If (New Object) then,
            Print “cache disk”
      Else
            Print “cache disk LFU”
    End If
```

Combination of FIFO-LRU algorithm: Cache has mechanism to determine whether a request for an object has been made previously. If it has ben made before, it means the requested object is stored and the cache and cache hit happens, otherwise cache miss hapens and the system must continue the search in the network. The combined algorithm FIFO-LRU does not discard objects although they are expired. On the contrary, these object will be queued for use. When there is a new object that needs to be stored in cache, the algorithm will detect the queued, expired objects. The pseudo-code is illustrated in Algorithm 4.

Algorithm 4; Pseudo code for combination FIFO-LRU:

```
Read Xn; Yn; Zn
    If (disk cache full≥cacheSize) then,
        Print "cache disk
    If (counter≥MaxCount; expires≥MaxTime) then,
        Print "cache disk"
      Else
        Print "cache disk FIFO-LRU"
    End If
```

Combination of FIFO-LFU algorithm: The combined FIFO-LFU cache replacement algorithm has similar principle to FIFO-LRU, except that it is fetch count that is used for checking, instead of expiry time as in FIFO-LRU. This is illustrated by pseudo-code in Algorithm 5. Objects that have smallest fetch count in the cache will be placed back in queue and not discarded directly.

Algorithm 5; Pseudo code for combination FIFO-LFU:

```
Read Xn; Yn; Zn
    If (disk cache full≥cacheSize) then,
            Print "cache disk
    If (counter≥MaxCount; fetch count≥MaxTime) then,
            Print "cache disk"
        Else
            Print "cache disk FIFO-LFU"
     End If
```

Combination of LRU-LFU algorithm: In the combined LRU-LFU algorithm, the expired objects are evaluated again based on fetch count as illustrated by pseudo-code in Algorithm 6.

Algorithm 6; Pseudo code for combination LRU-LFU:

```
Read Xn; Yn; Zn
    If (disk cache full≥cacheSize) then,
            Print "cache disk
    If (fetch count≥expires) then,
            Print "cache disk"
      Else
            Print "cache disk LRU-LFU"
    End If
```

If the objects are not accessed anymore, deletion or replacement of the objects is carried out.

Combination of FIFO-LRU-LFU algorithm: Pseudo-code for the combined FIFO-LRU-LFU algorithm is given in Algorithm 7. The algorithm replace an object that fulfills three criteria: it has been stored for the minimum of 5 days, its fetch count is 10 or more and it is the oldest incoming object. Accordingly, in the pseudo-code, cache replacement is done by examining three conditions, namely: fetch count≥max count; expires≥max time; max count≥max time.

Algorithm 7; Pseudo code for combination FIFO-LRU-LFU:

```
Read Xn; Yn; Zn
    If (disk cache full≥cacheSize) then,
            Print "cache disk
    If (fetchcount≥MaxCount; expires≥MaxTime;
            MaxCount≥MaxTime ) then,
            Print "cache disk"
      Else
            Print "cache disk FIFO-LRU-LFU"
    End If
```

For all of the above algorithms, cache size, max count, dan max time are the parameters. By combining algorithms, the deletion of an object is based on more than one criteria corresponding to the above three parameters. The performance test and measurement of all algorithms is done by realizing a system consisting of an ADSL modem connected to the internet, a server, a switch and three computers connected to the switch as illustrated in Fig. 3.

In the examination, requested objects from the three computers are defined as Xn, Yn dn Zn for each algorithm. After measurement system has been installed, different algorithms as well as data input are synchronized. After that, measurement of four parameters, i.e., hit rate, response time, throughput and delay time is made.

Measurement was made separately for different types of algorithm, each time lasting for 2 months: the single algorithms from April-June 2015, continued with the double algorithms from July to August 2015 and the combination of three algorithms, FIFO-LRU-LFU from October-December, 2015. During the measurement of performance of FIFO-LRU-LFU algorithm, the number of clients decreased due to coincidence with school holiday

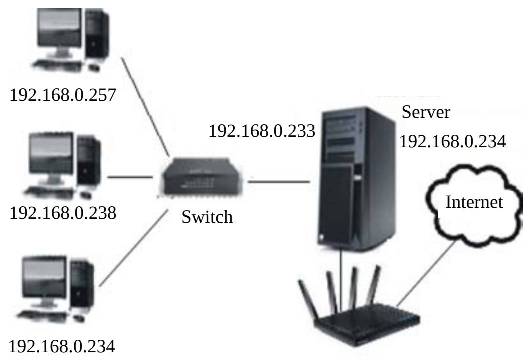

Fig. 3: Networking system configuration used for evaluation

and therefore, additional measurement was made from February-March, 2016. One day's measurement was taken from 5:00 p.m. 04:00 a.m. the next morning which yielded 15 records of cache entry per day. Subsequently, the most busy period was chosen, i.e., the period of time during which many clients were active.

Hit rate, response time, delay time and throughput were measured for all algorithms. Hit rate was measured by calculating the ratio between the number of object requests received and the number of objects sent by clients to cache. Response time is the period starting from a client sending a request for an object, regardless of its availability in the cache, until the client obtains the response for the request. Delay time is defined as total time of the process of searching object in the cache, calculated from the moment of object request until the search ends with either hit or miss. Throughput was measured as ratio of the number of objects given in bits experiencing hit to observation time.

In the experiment, requests were done through three laptops that were accessing internet at the same time for various objects Xn, Yn dan Zn. For that purpose, an open source software was installed at the server, client and firewall while the network was configured in order to enable.

The client computer to access applications in the server computer. The experiment reveals that the more clients accessing applications in the server simultaneously, the lower the execution speed.

Hit rate and response time as the performance indicators of the single algorithms, LRU, LFU dan FIFO as well as the double and the combination of three algorithms were evaluated by sending various previously specified objects to system request by using open source software Synchronization. The objects were then stored in the cache as illustrated in Fig. 3. In addition, cache object validation was done to ensure that expired data was not supplied to a client. The validation was done at the proxy server where it is checked that only if the copy of the object to be supplied to the client is still valid then the object is sent to the client.

RESULTS AND DISCUSSION

The combined FIFO-LRU-LFU algorithm indicates good and fast response, so that, hit rate percentage increases with cache size. The graphs of hit rate, delay time and througput with respect to cache size are illustrated in Fig. 4.

Figure 4a shows that hit rate increases with cache size up to cache size of about 100 MB, after which the value of hit rate is nearly constant. This can be explained as follows. The increase of number of requests that can be served well by the cache (i.e., hits) indicates the increase of percentage of object that is taken from the cache with respect to the total requests sent by clients to the cache. The highest hit rate is shown by FIFO-LRU-LFU algorithm which indicates the improvement of performance because

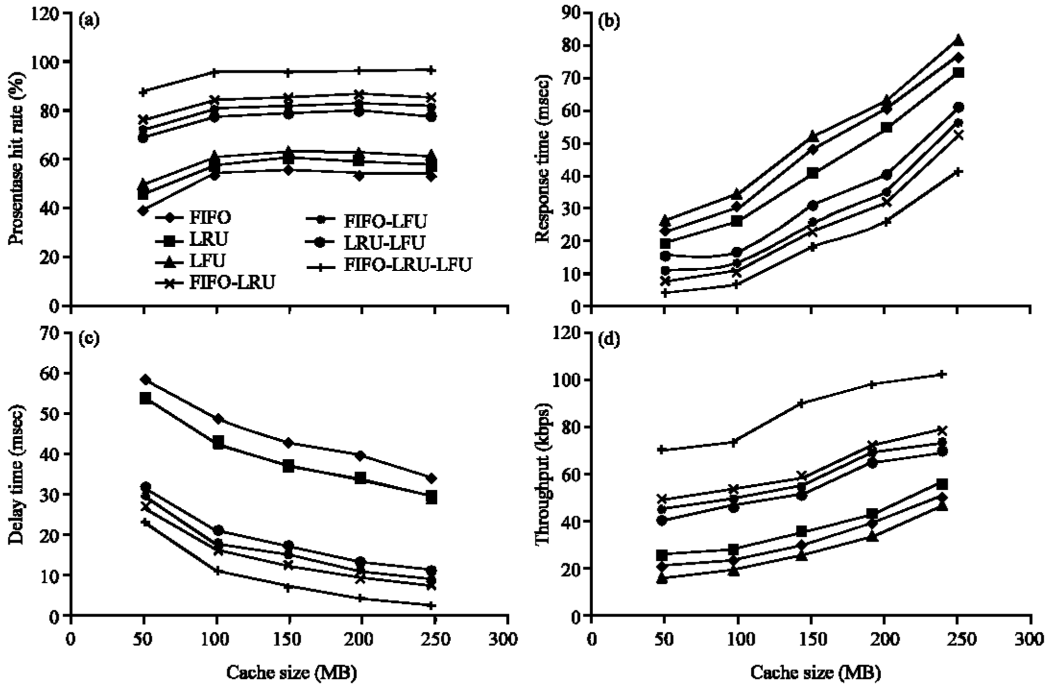

Fig. 4: Performance of cache replacement algorithms: a) Prosentase hit rate; b) Response time; c) Delay time and d) Throughput

a number of objects are not immediately deleted but instead are put back in queue which can increase hit opportunity. For cache size in the range of 100-250 MB, hit rate achieved with FIFO-LRU-LFU can reach 96%.

With regard to hit rate, based on measurement of object requests with cache size of 50 MB for FIFO-LRU-LFU algorithm, the percentage ratio of cache hit to the number of requests is 38% with FIFO, 45% with LRU and 47% with LFU. On the other hand, the FIFO-LRU yields hit rate percentage of 74% which increases up to 87% when the FIFO-LRU-LFU algorithm is used. As shown in Fig. 4b response time increases with cache size. This phenomenon occurs because as the cache size becomes larger, longer time is needed to search requested objects. Data analysis for 50 MB cache size gives response time of 21 msec with FIFO, 19 msec with LRU and 22 msec with LFU. Further, by implementing combination of two algorithms, the response time achieved with FIFO-LRU is 9, 11 msec with FIFO-LFU and 13 msec with LRU-LFU. With FIFO-LRU-LFU algorithm, the response time is only 4 msec, much lower than those of other algorithms.

Figure 4c indicates that with 50 MB cache, delay time caused by the use of any of the three single algorithms ranges from 54-59 msec. It reduces to 26-32 msec when any of the double-combined algorithms is used and goes further down to 23 msec when the FIFO-LRU-LFU is employed. Likewise with 150 MB cache, the delay time is in the range of 37-43 msec for single algorithms, 12-16 for the double-combined algorithms and only 7 msec for FIFO-LRU-LFU. Thus, there is a phenomenon that the bigger the cache size, the smaller the miss rate which results in faster object search and reduces the delay time.

The above results for response time and delay time seem to be contradictory, particularly on their variation with cache size. The following discussion explains why the two results differ. Delay time decreases when combination of FIFO-LRU-LFU is implemented as indicated in Fig. 4c. With bigger cache, longer time is needed to request objects. When an object request experiences a hit, the object is stored back in the cache disk. This reduces time delay when request of the same object is repeated. When the cache becomes full and combined FIFO-LRU-LFU is used in deletion of objects based on clock logic counter and stack of queued object, the delay time decreases even further with cache size.

Figure 4d indicates that in terms of throughput, FIFO-LRU-LFU algorithm is also better than other algorithms. With 50 MB cache, the throughput with single algorithm reaches 20-23 kbps. With combination of two algorithms there is an increase of throughput up to 40-45 kbps. Further with combination of three algorithms of FIFO-LRU-LFU the throughput is 66 kbps. With 150 MB cache, throughput increases with single algorithm to 28-32 kbps and with double algorithms up to 50-56 kbps. Throughput increases much greater with combination algorithm of FIFO-LRU-LFU which reaches 85 kbps because the greater chance for finding requested objects in cache will reduce access time to server that in turn increases throughput.

By comparing all the algorithms with respect to hit rate, response time, delay time and throughput, it is clear that LRU performs better than FIFO. For example, LRU yields response time 4 msec faster than FIFO for 100 MB cache and approximately 7 msec faster for 200 MB cache. This is because LRU maintains objects which are still often accessed stored in the cache, a thing that does not occur with FIFO when such objects have expired. Algorithm obtained from the combination of two single algorithms performs better than any of the single algorithm. FIFO-LRU is the best algorithm among other double combinations which achieves response time of 12 and 26 msec, respectively for cache size of 100 and 200 MB. With FIFO-LRU-LFU algorithm, there is much greater increase in performance of all criteria compared to either single or double algorithm. This is demonstrated by the delay time for FIFO-LRU-LFU being approximately 20 and 23 msec less than LRU for 100 and 200 MB cache, respectively and 4 and 8 msec less than FIFO-LRU for the same cache sizes.

When LRU is used which is based on stack and logic counter, leaps often happen during the process of object replacement in cache replacement which results in increase of delay. On the other hand when the basic principle of FIFO is concerned, earlier incoming objects will also come out earlier without considering whether these objects are still often requested. Therefore, when LRU is combined with FIFO, objects that have so far often experienced leap during the replacement will follow the queuing principle. When the FIFO-LRU-LFU algorithm is adopted, a number of hit requests for objects that have been stored in the cache disk does not require object search anymore. This causes object searching time to decrease which eventually results in better response time.

When the FIFO-LRU-LFU algorithm is used there is an increase of throughput of 95% relative to LRU with 100 MB cache and about 83% with 200 MB cache. The incease of throughput influences the bandwidth use efficiency and reduces delay time up to 25.6 and 11.8% with 100 and 200 MB cache, respectively, compared to delay time of LRU single algorithm.

CONCLUSION

The reported research proposes cache replacement algorithm which is the combination of three single algorithms LRU, LFU and FIFO. This combination is meant to blend constructively advantages of the different algorithms in order to improve performance of object access as measured by such indicators as hit rate, response time, delay time and throughput.

The result of experimental evaluation generally show that there is a relatively great increase in the object access performance as the result of combination of FIFO-LRU-LFU algorithm. When compared to FIFO-LRU algorithm, the percentage of hit rate from a number of object requests that can be supplied by cache using FIFO-LRU-LFU increases up to 83% when 200 MB cache is used and 95% with 100 MB cache. The increase of hit rate percentage increases request service speed which results in smaller response time. As a result, the delay time reduces with cache size which causes object searching time is shorter.

RECOMMENDATIONS

In addition with the increase of hit rate percentage, the time needed for searching objects requested repeatedly is smaller which eventually increases throughput. Based on the above observation, the combination of FIFO-LRU-LFU algorithm is strongly recommended to improve the throughput of client access.

REFERENCES

Aghaei, B. and N. Zaman-Zadeh, 2016. Evaluations of cache coherence protocols in terms of power and latency in multiprocessors. Asian J. Inf. Technol., 15: 5181-5186.

Ajorloo, H. and M.T. Manzuri-Shalmani, 2016. Throughput modeling of distributed reservation protocol. IEEE. Trans. Mob. Comput., 15: 503-515.

Akon, M., M.T. Islam, X. Shen and A. Singh, 2010. Hit optimal cache for wireless data access. Proceedings of the Conference on Global Telecommunications (GLOBECOM 2010), December 6-10, 2010, IEEE, Miami, Florida, USA. isBN:978-1-4244-5636-9, pp: 1-5.

Anandharaj, G. and R. Anitha, 2009. An improved architecture for complete cache management in mobile computing environments. Int. J. Soft Comput., 4: 142-147.

Chang, R.I., Y.L. Chen and Y.Y. Wu, 2008. Improving proxy cache performance by domain-behavior classification and on-demand caching. Asia J. Inform. Technol., 7: 100-108.

Hendrantoro, G. and A. Affandi, 2015. Early result from adaptive combination of LRU, LFU and FIFO to improve cache server performance in telecommunication network. Proceedings of the International Seminar on Intelligent Technology and its Applications (ISITIA), May 20-21, 2015, IEEE, Surabaya, Indonesia isBN:978-1-4799-7710-9, pp: 429-432.

Jacob, B., S. Ng and D. Wang, 2010. Memory Systems: Cache, DRAM, Disk. Morgan Kaufmann Publishers, Burlington, Massachusetts, USA. isB: 978-0-12-379751-3, Pages: 955.

Ji, M., G. Caire and A.F. Molisch, 2015. The throughput-outage tradeoff of wireless one-hop caching networks. IEEE. Trans. Inf. Theory, 61: 6833-6859.

Lee, D. and K.J. Kim, 2010. A study on improving web cache server performance using delayed caching. Proceedings of the International Conference on Information Science and Applications (ICISA), April 21-23, 2010, IEEE, Seoul, South Korea isBN:978-1-4244-5942-1, pp: 1-5.

Lenjani, M. and M.R. Hashemi, 2014. Tree-based scheme for reducing shared cache miss rate leveraging regional, statistical and temporal similarities. IET. Comput. Digital Tech., 8: 30-48.

Lim, K., Y. Bang, J. Sung and J.K.K. Rhee, 2014. Joint optimization of cache server deployment and request routing with cooperative content replication. Proceedings of the IEEE International Conference on Communications (ICC), June 10-14, 2014, IEEE, Sydney, New South Wales, Australia isBN:978-1-4799-2003-7, pp: 1790-1795.

Liu, Z., K. Zheng and B. Liu, 2007. Hybrid cache architecture for high-speed packet processing. IET. Comput. Digital Tech., 1: 105-112.

Meira, W. Jr., E. Fonseca, C. Murla and V. Almeida, 1998. Analyzing performance of cache server hierarchies. Proceedings of the 18th International Conference Chilean Society Symposium Society, (ICCSSS`98), IEEE Catalog, pp: 113-121.

Nakamura , H., M. Kondo, T. Ohneda, M. Fujita and S. Chiba *et al*., 2002. Architecture and compiler co-optimization for high performance computing. Proceedings of the International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems, January 11, 2002, IEEE, Big Island, Hawaii, USA. isBN:0-7695-1635-1, pp: 50-56.

Shim, J., P. Scheuermann and R. Vingralek, 1999. Proxy cache algorithms: Design, implementation and performance. IEEE. Trans. Knowl. Data Eng., 11: 549-562.

Stallings, W., 2000. Computer Organization and Architecture: Designing for Performance. 8th Edn., Pearson Education, India is BN:978-81-317-3245-8, Pages: 781.

Tanwir, G. Hendrantoro and A. Affandi, 2017. Combination of FIFO-LRU cache replacement algorithms on proxy serverto improve speed of response to object requests from clients. ARPN. J. Eng. Appl. Sci., 12: 710-715.

Vladimir, N.S., P.V. Andrey and C.G. Roman, 2017. Containerized mikroservice architecture performance increasing by using cache subsystem and multithreaded application server. J. Eng. Appl. Sci., 12: 1250-1253.

Wang, B., S. Sen, M. Adler and D. Towsley, 2004. Optimal proxy cache allocation for efficient streaming media distribution. IEEE. Trans. Multimedia, 6: 366-374.

Zhang, Q., Z. Xiang, W. Zhu and L. Gao, 2004. Cost-based cache replacement and server selection for multimedia proxy across wireless Internet. IEEE. Trans. Multimedia, 6: 587-598.