The RAM-based Web Proxy Servers

KIN-YEUNG WONG and KA-CHON LAI
Computing Program
Macau Polytechnic Institute
Rua Luis Gonzaga Gomes
MACAU
kywong@ipm.edu.mo chonchondotcom@gmail.com

Abstract: - The conventional web proxy servers use hard disk as their primary cache storage. However, as disks use mechanical operations, they are typically the slowest component of the servers. Though, in the last decade, various solutions have been proposed to improve the proxy performance, disks are still the performance bottleneck. Consider that the mainstream operating systems are 64-bit and the cost of memory has become economical in recent years. In this paper, a proxy server using main memory as the primary cache storage is proposed. Analytical results show that the ram-based proxy servers effectively eliminate the performance bottleneck of disk, and allow systems to scale well for higher browser rates.

Key-Words: - Web Proxy, Web caching, Web Traffic, Storage Design, Performance Evaluation

1 Introduction

A web cache server [1] (also called a proxy server or simply a proxy) is typically located near its clients. It stores the retrieved object (from a remote server) on a local disk, so that it is not necessary to contact the remote server next time its clients request the same object. The effective use of proxy not only shortens user perceived latency, but also reduces the number of requests reaching the remote servers, thus reducing overall network activity.

However, the web traffic is continuously growing in the last decade. The growth drivers include the increasing popularity of mobile devices [2][3] and wireless broadband [4] as well as the increasing number of Internet users in developing countries. As a result, proxies under overloaded condition commonly occur. As all web requests (from all clients) will be sent to their proxy server, if the server is overloaded, it will cause performance problems, including long delay or connection time-outs.

1.1 Disks as the performance bottleneck

The operations of a proxy are simple: It receives and processes requests from its clients, and then, if it finds the requested object in its local disk (a cache hit occurs), it returns the object to the user directly. Otherwise (a cache miss occurs), it retrieves the requested object from the remote origin server, and then stores the retrieved object into the local disk.

As can be seen in the operations, a proxy server performs disk I/O heavily. However, the performance of disk is limited by many factors: mechanical delay of a disk arm, command and protocol overheads on an I/O channel and logical-to-physical data block mapping. Therefore, disk is typically the slowest component in a proxy, making it the performance bottleneck. In other words, disks play a critical role in the performance of a proxy.

Proxy servers generally have a hit ratio of 20-40% depending on the traffic characteristic, which implies that 60-80% requests will cause cache miss. Serving cache-miss requests require file deletion and creation operations: first, a file of an old cached object will be removed from the disk (if a cache replacement is needed), and then, a file for the newly retrieved object has to be created.

Serving cache-hit requests primarily involves disk operations (mainly disk reads), and these operations contribute the largest portion of the request response time. On the other hand, though cache-miss requests require network operations, disk operations (mainly disk write) are necessary. Therefore, for both cache hit and miss situations, if the overhead caused by the disk can be avoided, the overall request response time can be reduced substantially.

1.2 Mitigation to disk performance problems

To solve the performance problems caused by disk, various mitigation approaches have been proposed.

1.2.1 Managements of cached objects in disk

One approach is to make the cached objects better organized in the disk. The traditional method uses direct mapping: it is to map a URL to a file system path directly. This simple method could cause a very long pathname depending on the lengths of URLs, and hence looking up a cache file from the disk can be time consuming (involving a number of i-node search).

Today, the common method uses hash mapping, which is adopted by the popular open source proxy server Squid Proxy [5]. It is to create a fixed cache directory structure beforehand (typically, three levels), and the cached objects are stored in the lowest level. The location of a cached object is determined by the hash value of its URL. Due to the nature of hash functions, each directory would has the similar number of cached objects, which makes the directory traverse time relatively short and stable.

Wong and Yeung further improve the disk access performance by using site-based mapping [6]. It places files belonging to a given site in the same file directory. This will cause file systems (such as fast file system and ext2) to store web objects belonging to the same site in nearby disk blocks. As a result, it would reduce I/Os for the disk head when viewing a web page, because its related files (such as embedded images) are stored likely in the adjacent disk blocks. The simulation results show that it reduces disk access time by 21–50%, compared to the conventional hash mapping method.

Other proposed methods include the work trying to reduce the disk I/O of web proxy server in [7], the work using the web-conscious storage management in [8], as well as the scheme using the efficient management for the proxy cache objects in [9].

1.2.2 File systems

Another approach is to use a specialized file system. Traditional file systems are optimized for general-purpose usage pattern, whereas Web proxies have a different usage pattern. For example, the read operations more than write operations in traditional file systems, whereas web access is a write-dominated usage pattern. Wang et al. [10] proposed a customized file system, called UCFS, to improve the disk I/O performance of proxy servers. UCFS uses efficient meta-data tables to eliminate almost all I/O overhead of meta-data searches and updates. It also uses large disk transfers to significantly improve disk write performance. Results of simulation using real-world access traces shows that ICFS achieves 8-19 times better I/O performance than the Unix Fast File System (FFS).

On the other hand, Shriver et al. [11] proposed a light-weight file system called Hummingbird for Web proxy, which separates object naming and storage locality through direct application-provided hints. Their experimental results show that Hummingbird achieves document request throughput 2.3 to 9.4 times larger than that by the Unix File System. Other techniques to optimize the file system for web objects include [12][13].

1.2.3 Others

Besides the above two approaches, Wong and Yeung proposed a novel approach to reduce the disk load of a proxy server [6][14]. The approach is to forward only those requests to popular web sites (called hot requests) to the proxy server, and forward others (called cold requests) to the remote servers directly bypassing the proxy server. Since this approach is able to prevent the proxy from processing the unnecessary cold requests, the disk overhead caused by those requests can be eliminated.

1.3 Use main memory as the primary storage

As can be seen, many previous attempts are made to mitigate the performance problems caused by disks. In this paper, we argue that if the use of disks can be avoided, the problems associated with disks can be eliminated, leading to substantial improvement on the overall performance.

Instead of using disk as the primary cache storage, we propose the use of main memory. While it is obvious that the performance of RAM is faster than that of a disk, the use of RAM as the primary storage is not ready until recent years:

1. Since 64-bit CPU has become mainstream and most modern operating systems have already become 64-bit ready, supporting up to a theoretical 16TB of RAM. With the support of such huge capacity, software designers are free to design software that can store as much useful data as possible in memory to achieve higher performance.
2. Since the densities of chip increase, memory can store huge data which makes the capacity and price of main memory can follow the traditional disk.

On the other hand, the recent breakthroughs of memory technologies also support the feasibilities of ram-based storage in the design of future proxy servers. For example, IBM and Micron Technology announced the production of a new memory device that uses through-silicon vias (TSVs) [15], which enable memory chips to be built in three dimensions. As a result, the new memory chips can achieve 128 Gbytes per second (GB/s) bandwidth, while the current high-level Dynamic RAM (DRAM) delivers around 12.8 GB/s. On the other hand, IBM researchers have successfully created a single data bit by using only 12 atoms [16]. This achievement points toward magnetic storage that is 417 times denser than today's DRAM.

The conventional design and implementation of proxy servers are based on the assumption that the disk is the primary cache storage. Putting massive amount of RAM into the machine that is running the conventional server software could achieve higher performance. However, in this paper, we argue that if there is sufficient amount of RAM, to optimize performance, the server should be designed and implemented with the notion that RAM is used as the primary storage. We call this type of server RAM-based server.

This paper is organized as follows. Section 1 reviews the performance problems caused by hard disks, and most importantly, points out that the use of as the primary cache storage. Section 2 presents the characteristics and a design of ram-based web proxy servers. Section 3 analyzes the performance of the ram-based server, and the corresponding numerical and simulation results are shown in Section 4. Finally, section 5 presents the conclusion.

2 Designing a RAM-Based Web Cache Server

2.1 Use main memory as the primary storage

We refer the *disk-based design* to those designs with the notion that disk is the main storage and using disk I/O system calls (e.g., fopen() in the C programing language) in their implementation for retrieving the objects in the disk. This notion will not be changed even though the server is equipped with numerous main memory.

We refer the *RAM-based design* to those designs with the notion that memory is the main storage and directly retrieving the objects in memory through the manipulation of data structures (e.g., array and link disk).

In this section, we will first review the conventional (disk-based) cache architectures, and then demonstrate how to only the operations can be designed merely using RAM as the main storage, called ram-based.

The following reviews the differences between RAM and disk:

1) RAM consists of chips that store data electronically whereas disk consists of platters that store data magnetically. Therefore, the data stored in RAM is volatile whereas those stored in disk is permanent.
2) If the data requested is stored in disk, a copy of it will be temporarily put into RAM before being processed by CPU. Therefore, if the requested data is already in RAM, the access time is much shorter.
3) Random access in disk is slower than sequential access because the former requires more disk mechanical movements, whereas the performances of these two accesses are similar in memory. Therefore, the layout of data in memory is less critical than that on a disk.

The differences 2) and 3) point out the advantages of RAM-based servers, whereas the difference 1) points out the limitation which will be addressed in the conclusion section.

2.2 Conventional disk-based proxy design

2.2.1 Cache architecture

Fig. 1 shows the typical design of the object management in the conventional (disk-based) proxy servers. A fixed cache directory structure is created beforehand (typically, three levels). The cached objects are on the lowest level, and the location of a cached object is based on the hash value of its URL.

In Squid, a widely used open source proxy server, the default configuration is that the number of first-level directories under the cache root is 16, and the number of second-level directories under each first-level file directory is 256. Cached files will then be evenly mapped to the second-level subdirectories.

The metadata of each object in the disk is stored in a data structure, called StoreEntry. A hash table is used to store all StoreEntrys. A StoreEntry contains two pointers: sfileno and sdirno. sdirno points out the path of the object in the disk and sfileno is the actual file name of the object.

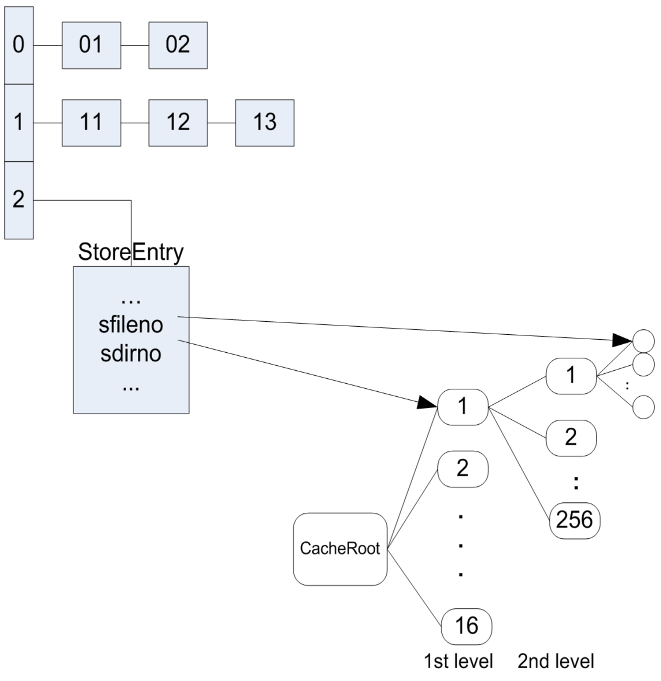


Fig. 1 Typical object management of the disk-based design

As can be seen, in the disk-based proxy server, the memory is used to store the metadata, the hash table of StoreEntry, whereas the disk is used to store the actual objects.

2.2.2 Cache Miss

The operations for a cache miss are as follow and shown in Fig. 2:

1. Upon receiving the request from a client, it searches for the corresponding StoreEntry in the hash table.
2. As it cannot find the object, it contacts the remote origin server for the requested object on behalf of the client.
3. After receiving the object from the origin server, it will create a StoreEntry to store the metadata of the object and add it into the hash table.
4. Based on the metadata specified in its StoreEntry, it stores the actual object in the corresponding file folder in the disk. Meanwhile, it returns the object to the client.

As can be seen, the process mainly requires two memory operations (steps 1 and 3) and one disk operation (step 4).

2.2.3 Cache Hit

The operations for a cache miss are as follow:

1. Upon receiving the request from a client, it searches the corresponding StoreEntry in the hash table.
2. It examines the metadata (sfileno and sdirno) stored in the StoreEntry to determine the disk location of the object.
3. It retrieves the actual cached object from the disk and returns it to the client.

As can be seen, the process mainly requires also two memory operations (steps 1 and 2) and one disk operation (step 3).

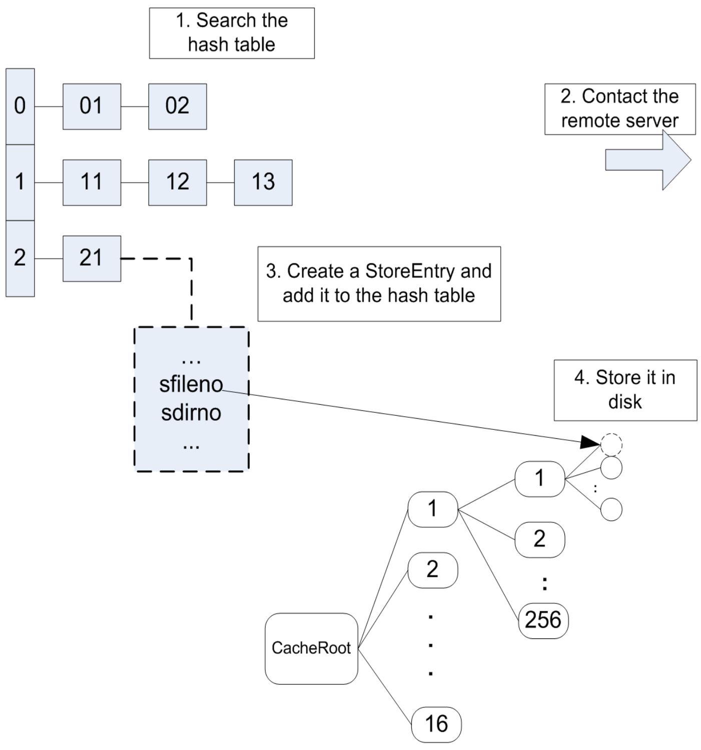


Fig. 2 Operation of Cache Miss in Disk-based Proxy

2.3 RAM-based design

As can be seen in the conventional design shown in Section 2.2, the disk-based design uses a hash table (memory) to identify the existence of an object, and uses a StoreEntry to determine the disk location of the object. In the RAM-based server design, this process can be simplified.

2.3.1 Cache Architecture

As we have sufficient memory space, we can use a simpler data structure to manage the cached objects in the memory, as shown in Fig. 3. A hash table can be used to link up all the cached objects directly in the memory. Each actual object is added by a header that stores all the necessary metadata (e.g., URL, timestamp, etc) for the cache operations. The object header is a fixed length data structure whereas the size of the actual object varies.

The advantage of such design is that it can reduce the number of memory reference. Unlike the disk-based operations that require another data structure (StoreEntry) to identify the disk location of a cached object, this design can retrieve the object immediately after the hash table lookup.

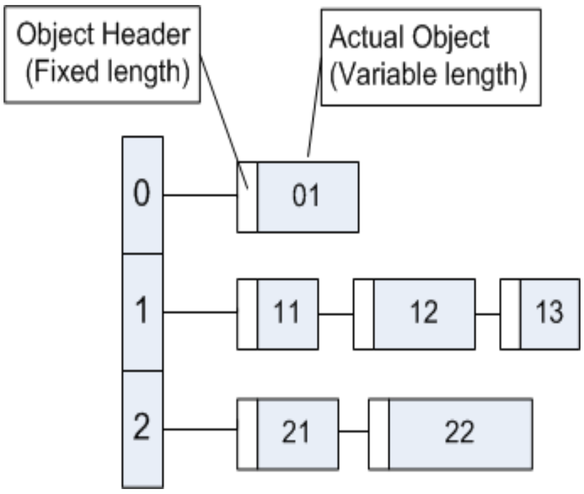


Fig. 3 Object management of the RAM-based proxy

2.3.2 Cache Miss

By using memory as the cache storage, the operations for a cache miss become:

1. Upon receiving the request from a client, it searches the hash table for the requested object.
2. As it cannot find the object, it contacts the remote origin server for the requested object on behalf of the client.
3. After receiving the object from the origin server, it will add it into the hash table. Meanwhile, it returns the object to the client.

2.3.3 Cache Hit

The operations for a cache hit are as follow:

1. Upon receiving the request from a client, the proxy will firstly lookup the reference of the information of the object in the hash table.
2. After that, it will retrieve the real object in the RAM and return it to the client.

2.3.3 Simplified Operations

As can be seen, for cache miss cases, the step 3 in the process using the RAM-based design replaces the steps 3 and 4 in the process using the conventional design (as described in Section 2.2.2). Similarly, for cache hit cases, the step 1 replaces the steps 1 and 2 in the process using the conventional design (as described in Section 2.2.3), and the step 2 involve memory operation instead of disk operations. These designs not only simplify the proxy operations but also reduce the workload offered to the proxy as the disk is bypassed.

3 Performance Analysis

Consider a typical local area network in which there are a number of client machines connecting to a proxy server, as shown in Fig. 4. Web requests will be sent to the proxy which in turn forward the requests to the remote web servers in the Internet. The proxy acts as a cache server storing frequently requested web objects locally.

We then analyze the response time of the conventional disk-based proxy and show how it can be improved if ram-based design is used in the numerical results section.

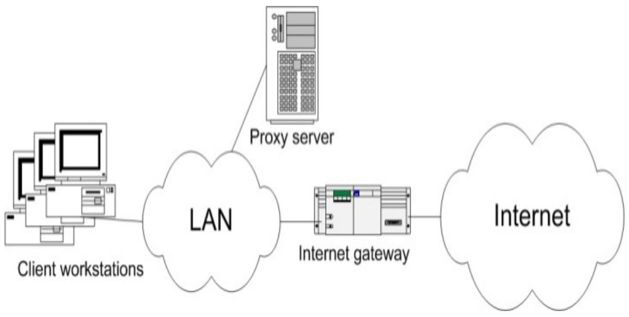


Fig. 4 A typical network connected to the Internet

3.1 Estimated Proxy Load Reduction

Suppose that there are N requests arrived to the proxy, r_1, r_2, …, r_N, for the web objects o_1, o_2, … o_N, respectively.

When the proxy receives r_i from a client, if o_i is in its local cache storage, a cache hit occurs and the proxy will respond the client with o_i directly from its local storage. If otherwise, a cache miss occurs and the proxy will make another request to the remote server for o_i.

Proxy hit ratio is defined as:

$$H = \frac{\text{total number of hits}}{N}$$

When a cache hit occurs, let L be the amount of the processing workload and let D be the amount of disk workload offered to the proxy. On the other hand, when a cache miss occurs, the proxy needs to perform an extra work of making another request r_i to the remote server. By using a similar approach for load estimation as suggested in [17], we assume that the extra processing workload offered to the proxy is also equal to L, giving a total of $2L$. For simplicity, we assume the amount of disk workload is also D.

As a result, the load offered to the disk-based proxy can be obtained by:

$$Load_{conventional} = H(L + D) + (1 - H)(2L + D)$$

Since the ram-based proxy server uses memory (instead of disk) for the cache storage, define M is the corresponding load for the proxy for either a cache hit or miss. Similarly, the load offered to the ram-based proxy can be obtained by:

$$Load_{ram} = H(L + M) + (1 - H)(2L + M)$$

Therefore, the estimated load reduction using the ram-based proxy can be obtained by:

$$\begin{aligned} LR &= \frac{Load_{conventional} - Load_{ram}}{Load_{conventional}} \\ &= \frac{D - M}{D + (2 - H)L} \\ &= \frac{D/M - 1}{D/M + (2 - H)\, L/M} \end{aligned}$$

As both M and L are much smaller than D, if we assume that they are similar and $L \approx M$, then:

$$LR \approx \frac{D/M - 1}{D/M + (2 - H)}$$

As $H<1$,

$$LR < \frac{D/M - 1}{D/M + 1}$$

It can be seen that the higher the ration of D/M, the higher the load reduction can be achieved. For example, for $D/M = 10$, the maximum load reduction that can be achieved is 9/11=82% whereas it can be 99/101=98% if D/M=100.

3.2 Modeling of Caching Systems

3.2.1 Queuing Network Model for the Conventional Proxy

As shown in the previous studies [18][19], the conventional proxy system can be modeled as a closed queuing network with eight queues (see Fig. 5):

clients:	the set of clients, where the delay is the average think time a client spent between making requests.
LAN:	the LAN connecting the clients and proxy.
router:	the Internet gateway with small latency.
outLink:	the transmitting link to the ISP.
Internet:	the total delay at the ISP, at the ISP's link to the Internet, at the Internet itself, and at the remote servers.
inLink:	the receiving link to the ISP.
CPU:	the CPU in the proxy.
disk:	the disk in the proxy.

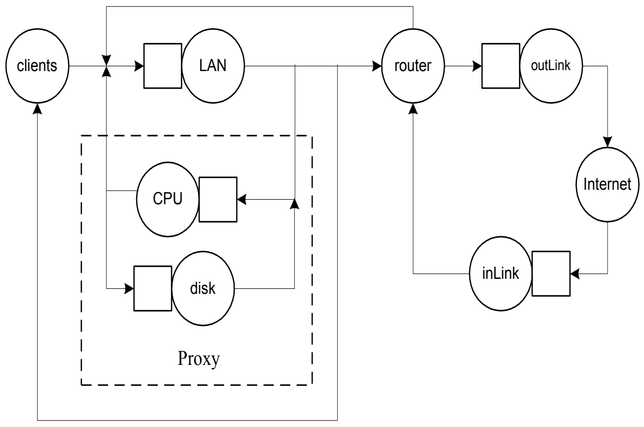


Fig. 5 The queuing network model of the conventional caching system

The following parameters have to be considered in this model:

EffectiveClients:	effective number of clients (client workstations) actively using the Web
BrowserRate (requests/sec):	Rate at which a browser requests a new object

LANBandwidth (Mbps):	LAN bandwidth
HTTPRequestSize (Bytes):	average size of the HTTP requests (generated by the browser)
DocumentSize (kBytes):	average size of all requested objects
Phit	fraction of requests that can be served from the proxy's cache
HitCPUTime (sec):	CPU time needed to process the tasks for a cache hit at the proxy server
MissCPUTime (sec):	CPU time needed to process the tasks for a cache miss at the proxy server
DiskTime (msec/kBytes):	disk time at the proxy
MaxPDU (Bytes):	maximum PDU size for the LAN's network layer protocol
RouterLatency (usec/datagram):	router latency per passing datagram
LinkBandwidth (kbps):	bandwidth of the connection to the that ISP
InternetRTDelay (msec):	Internet round trip delay between the ISP and the remote originating server
InternetDataRate (kBps):	Internet data transfer rate

3.2.2 Services Demands of the Queues

Before solving the queuing model, we need to calculate the service demands of the queues have to be obtained. The service demand of a queue is the sum of the service times at the queue over all visits to that queue. And the service times of the queues are calculated based on the parameters listed above.

Table 1 summarizes the services demands for the queues, where D_{LAN}, D_{router}, $D_{outLink}$, $D_{Internet}$ and D_{inLink} are the service demands for the queues in the model of a basic browser-server system without a proxy. The derivation of the equations of these service demands can be found in [18][19].

3.2.3 Modeling for the RAM-based Proxy

The modeling for the conventional disk-based proxy can be directly applied to that for RAM-based Proxy. The queuing model of the RAM-based proxy is same as the one shown in Fig.2 except that the disk queue should be replaced by a RAM queue.

On the other hand, one more parameter is needed: $RamTime$ (msec/Bytes) which is the average time spent in the RAM cache storage. With this parameter, the service demand for the RAM queue is $D_{RAM}^{proxy} = RamTime \times DocumentSize$.

Queue	Service demand
LAN	$D_{LAN}^{proxy} = Phit \times D_{LAN} + (1 - Phit) \times 2 \times D_{LAN} = (2 - Phit) \times D_{LAN}$
router	$D_{router}^{proxy} = (1 - Phit) \times D_{router}$
outLink	$D_{outLink}^{proxy} = (1 - Phit) \times D_{outLink}$
Internet	$D_{Internet}^{proxy} = (1 - Phit) \times D_{Internet}$
inLink	$D_{inLink}^{proxy} = (1 - Phit) \times D_{inLink}$
CPU	$D_{CPU}^{proxy} = Phit \times HitCPUTime + (1 - Phit) \times MissCPUTime$
Disk	$D_{disk}^{proxy} = DiskTime \times DocumentSize$

Table 1 The service demands for the queues in the model of conventional caching system

4 Numerical and Simulation Results

4.1 Performance Evaluation using MVA

The closed queuing network of the cache system shown in Fig. 5 can be solved by Mean Value Analysis (MVA) [20]. In order to examine the proxy performance improvement, we need to set the values for the model parameters, as shown in Table 2. By substituting the values to the service demand equations, the service demands for the two systems can be obtained. And, with the service demands, by using MVA, we can obtain the average request response times and the queue utilization for the caching systems. The results are shown in Fig. 6.

Considering the caching system with the disk-based proxy, as can be seen in Fig. 6(a), as browser rate increases, the disk utilization increases as well. When the browser rate reaches about 0.1 request/sec, the disk becomes 100% utilized, creating the performance bottleneck of the system, which in turn starts to limit the overall system throughput, as shown in Fig. 6(b).

Fig. 6(b) also shows that the system throughput for the ram-based proxy increases as the browser rates increases. It indicates that the ram-based proxy is able to eliminate the performance bottleneck of disk, allowing the system to scale well for higher browser rates. For example, as shown in Fig. 6(b), during the light load, the overall system throughputs for the two types of proxy servers are similar. However, as the browser rate increases the throughput of the disk-based proxy is limited to around 60 requests/sec due to the disk performance bottleneck as mentioned earlier. And when the browser rate reaches 0.4 request/sec, the system throughput for the RAM-based proxy can achieve the rate of 350 request/sec, whereas the disk-based proxy is still limited to 60 requests/sec.

HTTPRequestSize	300
HitCPUTime	0.25
MissCPUTime	0.50
MaxPDU	65,535
RouterLatency	50
LANBandwidth	100
InternetRTDelay	100
InternetDataRate	100
EffectiveClients	500
DocumentSize	30
Phit	0.4
DiskTime	0.5
RamTime	0.005

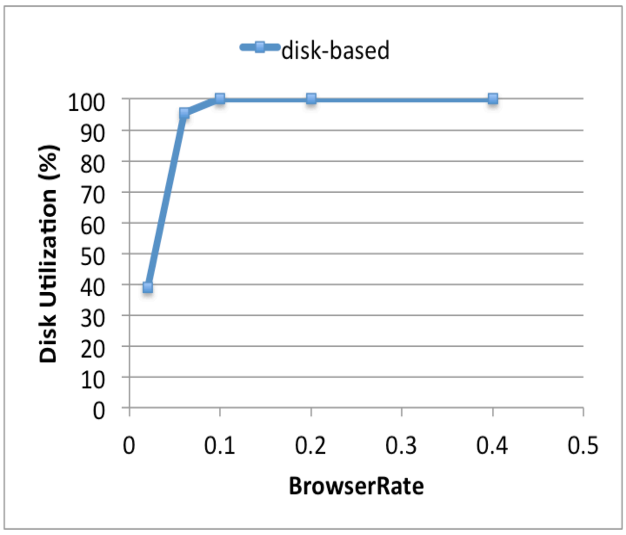
Table 2 The used parameter setting

4.2 Performance Evaluation using Simulation

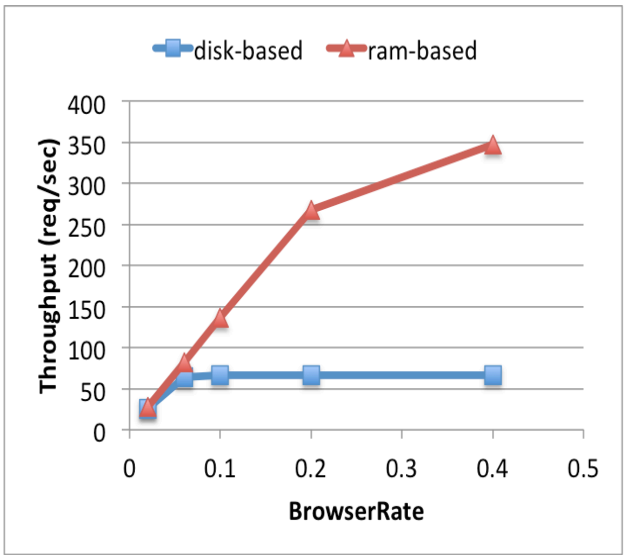
To evaluate the performance of RAM-based proxy, we configured the Squid proxy server to make it to use the main memory as the cache storage without the access to the disk. Our goal of this evaluation is to show that RAM-based proxy is feasible and provide significant performance improvements.

To measure the performance of the RAM-based proxy and compare it to that of the conventional disk-based proxy (also running Squid), we used the Web Polygraph proxy performance benchmark (version 4.3.2), which is a *de facto* industry standard for benchmarking proxy servers [21].

In the simulation tests, the Web Polygraph was configured to use the workload distribution parameters of the *Fourth Polygraph Cache-off* ("Polymix-4") [22], which simulates the real world web traffic. The workload schedule consists of 10 test phases, and each phase features different robot population size and offered load, which is to simulate the workload offered to the real-world proxy servers. The total time for one simulation test is about 10.33 hour.



(a) Disk Utilization



(b) System Throughput

Fig. 6 Analytical results

We used the 3.1 version of Squid proxy server, and ran it on a Linux machine with a 2.4 GHz CPU, 16 Gbytes Memory. It is also equipped with a 120 Gbytes disk. The cache storage size is 12 Gbytes for both RAM-based and disk-based proxy servers.
We used another two machines for the Web Polygraph Client (WP-Client) and Server (WP-Server) respectively. We connected these two machines and the proxy in a separated, independent network using a high-speed Ethernet switch so that the network bandwidth is not the bottleneck.

We performed experiments varying the offered load, so as to see the performances of the two types of proxy. Fig. 7 shows the results of the average response time for the hit requests of the two types of proxy servers as a function of the offered load (system throughput).

As can be seen in the figure, the performance of the disk-based proxy in terms of response time start to deteriorate quickly as the offered load reaches around 700 request/sec, at which the ram-based proxy is still providing a very low response time (about 3 msec). The throughput for the disk-based proxy is saturated at the offered load of around 800 request/sec, as at that rate, the response time reaches an unacceptable value, 517 msec. This result also supports that the ram-based proxy is able to eliminate the performance bottleneck in the disk-based proxy and allow the system to scale well for higher offered load.

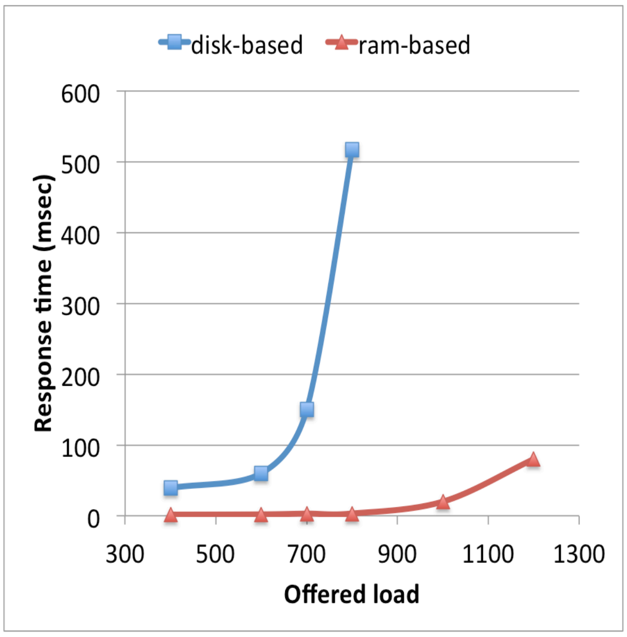


Fig. 7 Benchmark results

5 Conclusion

The conventional web proxy servers use disk as their primary cache storage. As disks involve mechanical operations, it is typically the slowest component in the proxy and the performance bottleneck. Whereas many previous efforts were proposed to reduce the disk load, this paper proposes another approach that is to use main memory as the primary cache storage.

In this paper, we have introduced the design of a RAM-based proxy. We have also used the closed queuing network to model the caching systems using a disk-based and a RAM-based proxy respectively. Analytical results show that the ram-based proxy effectively eliminates the performance bottleneck caused by disk, and allows system to scale well for higher browser rates. For example, when the browser rate reaches 0.4 request/sec, the system throughput for disk-based proxy is 60 requests/sec, whereas the RAM-based proxy can achieve the 350 request/sec.

However, the limitation of RAM-based proxy servers is that the objects stored in the cache are volatile. That is, unlike the disk-based proxy, if a RAM-based proxy is rebooted after a failure or system maintenance, the objects in the cache will not exist, and the cache has to go through a warm up process before achieving a reasonable hit ratio. Nonetheless, regularly backup the objects in the RAM to disk storage can relief the problem.

This paper applies the memory-based design to the web proxy server. Considering the advance of memory technologies and the cost of memory has become economical in recent years, it is anticipated that more and more types of servers and applications can make use of large amount of memory in their design to enhance the overall efficiency and performance.

Acknowledgment

The work described here is supported by Macao Polytechnic Institute Research Grant No. RP/ESAP-1/2009.

References:

[1] K.Y. Wong, Web Cache Replacement Policies: A Pragmatic Approach, *IEEE Network*, Vol.20, Iss.2, 2006, pp. 28-34.

[2] Angus K.Y. Wong, Cell Phones as Mobile Computing Devices, *IT Professional*, Vol. 12, No. 3, May/June 2010, pp. 40-45.

[3] I. S. Lei and K. Y. Wong, The Multiple-Touch User Interface Revolution, *IT Professional*, Vol. 11, No. 1, Jan./Feb. 2009, pp. 42-49.

[4] Angus K.Y. Wong, The Near-Me Area Network, *IEEE Internet Computing*, vol. 14, no. 2, Mar./Apr. 2010, pp. 74-77.

[5] Squid Caching Proxy, http://www.squid-cache.org/.

[6] K.Y. Wong and K. H. Yeung, An Alternative Web Caching Design: a Site-based Approach,

IET Communications, Vol.4, iss.12, 2010, pp. 1504-1515.

[7] C. Maltzahn, K. J. Richardson and D. Grunwald, Reducing the disk I/O of web proxy server caches, *Proc. USENIX Ann. Technical Conf.* (USENIX-99), 1999, pp. 225-238.

[8] E.P. Markatos, D.N. Pnevmatikatos, M.D. Flouris and M.G.H. Katevenis, Web-conscious storage management for web proxies, *IEEE Trans. Networking*, 2002, pp. 735-748.

[9] K. Cheng, Y. Kambayashi and M. Mohania, Efficient management of data in proxy cache, *Proceedings of 12th International Workshop on Database and Expert Systems Applications*, 2001, pp. 479-483.

[10] J. Wang, R. Min, Y. Zhu, and Y. Hu, UCFS - a novel User-space, high performance, Customized File System for Web proxy servers, *IEEE Transactions on Computers*, Vol.51, Iss.9, 2002, pp. 1056-1073.

[11] E. Shriver, E. Gabber, L. Huang and C. Stein, Storage Management for Web Proxies, *USENIX Annual Technical Conference*, 2001, pp. 203-216.

[12] P. Lensing, D. Meister and A. Brinkmann, hashFS: Applying Hashing to Optimize File Systems for Small File Reads, *International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI)*, 2010, pp. 33-42.

[13] L. Yu, G. Chen, W. Wang and J. Dong, MSFSS: A Storage System for Mass Small Files, *Proc. 11th International Conference on Computer Supported Cooperative Work in Design*, 2007, pp. 1087-1092.

[14] K. Y. Wong and K. H. Yeung, Site-Based Approach in Web Caching Design, *IEEE Internet Computing*, Vol.5, No.5, 2001, pp.28-34.

[15] 3D Chip-Making Capability, http://www.ibm.com/news/ae/en/2011/12/04/s971220m34896l98.html.

[16] Structured Memories, *Science*, Vol. 335, Iss. 6065, 2012, pp. 144-a

[17] K. Y. Wong and K. H. Yeung, A Dispatching Technique to Solve the Overloading Conditions of Web Cache Servers, *WSEAS Transaction on Communications*, Vol. 5, Iss. 5, 2006, pp.725-731.

[18] D. A. Menasce, L. W. Dowdy, V. Almeida, *Performance by Design: Computer Capacity Planning By Example*, Prentice Hall, 2005.

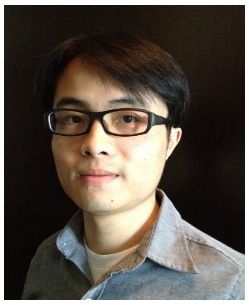
[19] K. Y. Wong and K. H. Yeung, Analytical Study on Web Caching Systems using Closed Queuing Network Modeling, *WSEAS Transaction on Communications*, Vol.5, Iss.5, 2006, pp.732-737.

[20] M. Reiser and S. S. Lavenberg, Mean-Value Analysis of Closed Multichain Queuing Networks, *Journal of the ACM,* Vol.27, Iss.2, 1980, pp. 313-322.

[21] Web Polygraph. http://www.web-polygraph.org/.

[22] PolyMix-4 as Web traffic workload. http://www.web-polygraph.org/docs/workloads/polymix-4/

Angus K.Y. Wong received his B.Sc. and Ph.D degrees in information technology from the City University of Hong Kong in 1998 and 2002 respectively. Wong is currently a Professor at Macao Polytechnic Institute. His research interests include Internet systems, network infrastructure security, and mobile computing. He is also an active industry consultant in the areas of computer networking and communication systems. Wong and Yeung co-authored Network Infrastructure Security, Springer, 2009.



Ray K.C. Lai received the B.Sc. degree in Computer Studies from Macau Polytechnic Institute in 2009, and is currently working toward the M.Sc. degree in Software Engineering from University of Macau. His research interests include Internet systems and networking.