Web Cache Replacement Policies: A Pragmatic Approach

Kin-Yeung Wong, Macao Polytechnic Institute

Abstract

Research involving Web cache replacement policy has been active for at least a decade. In this article we would like to claim that there is a sufficient number of good policies, and further proposals would only produce minute improvements. We argue that the focus should be fitness for purpose rather than proposing any new policies. Up to now, almost all policies were purported to perform better than others, creating confusion as to which policy should be used. Actually, a policy only performs well in certain environments. Therefore, the goal of this article is to identify the appropriate policies for proxies with different characteristics, such as proxies with a small cache, limited bandwidth, and limited processing power, as well as suggest policies for different types of proxies, such as ISP-level and root-level proxies.

Web caching is a significant part of today's Web infrastructure. It is important because it reduces user-perceived retrieval latencies, the number of requests reaching Web servers, and overall network activity. Because of this, Web cache servers (aka proxy servers) are widely deployed in many places throughout the Web. In general, it is typically located at the boundary of a network intercepting all Web requests. The basic operation of a proxy is shown in Fig. 1.

When Luotonen and Altis [1] first proposed the proxy server in 1994, Web cache replacement policy was already a hot research topic. Such a policy is used to manage cache content. In the beginning, traditional replacement policies such as least recently used (LRU) and least frequently used (LFU) were employed. Later, various kinds of new replacement policies were proposed. At the time of writing, at least 50 policies have been reported in the literature (Table 1b). Considering that there is a multitude of replacement policies, research on more policies is less important. As a matter of fact, recently proposed solutions provide only slight improvements. We believe we already have sufficient good policies for usage.

Indeed, most of the policies in various proposals provide evidence that they perform better than others. This leads to a state of confusion as to which policy should be used. Actually, there is no single policy that performs best in all environments. This is because different policies have different design rationales and are designed to optimize different resources. The goal of this article is to suggest suitable policies for use in different environments based on their characteristics, rather than propose another new policy to challenge the existing ones.

Our proposal takes a different approach, providing instead useful information for policy selection. Although Web cache replacement policies have been summarized in previous work, from small overviews [2, 3] to more comprehensive surveys [4, 5], many of them focus on discussing the operations of common policies and comparing their performance by running trace-driven simulations. Instead, we aim to provide practical information for usage purposes. For example, we point out which kinds of systems should be concerned with which performance metric (Table 2). We also point out the design rationale and characteristics of different categories of policy (Table 1). More important, we provide usage suggestions about which policy is appropriate for which particular environment (Table 3). This information is useful to cache designers, developers, administrators, and end users, and can be summed up as a "fitness for purpose" approach.

Replacement Policy Review

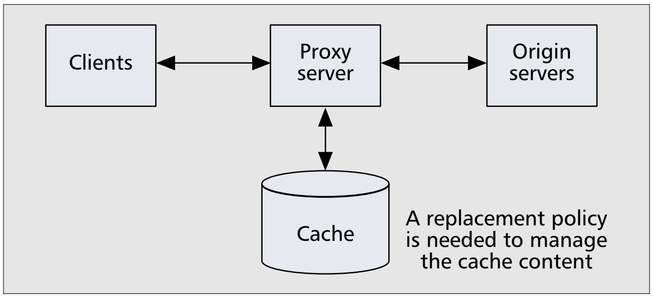
Replacement Policy Basics

A cache server stores Web objects (e.g., HTML pages, images, and files) locally for the use of future requests to those objects. As cache size is finite, a cache replacement policy is needed to manage cache content. If a cache is full when an object needs to be stored, the policy will determine which object is evicted to make room for the new object. However, in practical implementation a replacement policy usually takes place before the cache is really full. The cache uses two watermarks, *high* and *low*, to guide the replacement process. If the size of total cached objects exceeds the high watermark, the policy will evict objects until the low watermark is reached. The advantage of doing this is reducing the overhead of invoking the policy on demand.

The goal of the replacement policy is to make the best use of available resources, including disk space, processing power, server load, and network bandwidth. Some policies favor one resource at the expense of another. Later in this section we show which kind of policy makes good use of which resource and should be used in which environment.

Note that in this article we only consider the cache replacement policy for cache servers located between clients and origin servers, acting as a proxy. There are caches placed directly in front of a particular origin Web server (called Web server caches or httpd accelerators). Since a Web server cache serves its own Web server only, it knows the information (e.g., object popularity, average object size, and number of total objects)

0890-8044/06/$20.00 © 2006 IEEE



■ Figure 1. *Web proxy servers temporarily store retrieved Web objects in their caches, so it is not necessary to contact the origin servers the next time objects are requested by their clients.*

on the objects it is going to cache. This makes the design of a replacement policy for a Web server cache different from one for a proxy cache.

Performance metrics are used to evaluate a policy. It is not surprising that a policy performs very well in terms of one performance metric but poorly in terms of another. This is because some metrics are not achieved at the same time. Which metric we should be concerned with depends mostly on the environment in which the policy is being used. We discuss the following common metrics. Table 2 shows the environments to which the metrics are of particular interests.

Hit Rate — As shown in Fig. 1, if the proxy finds the requested object in its local cache, it returns the object to the user directly without contacting the origin servers. Hit rate (HR) is defined as the percentage of requests that can be satisfied by the cache. For example, an HR of 60 percent indicates that six of every 10 requested objects can be found locally in the cache. Note that HR only indicates how many requests are hits and does not indicate how much bandwidth or latency has been saved. Nevertheless, it is a very good performance indicator and, in most cases, is the first metric to be considered. It is of interest to systems with small cache size because most policies provide similar HRs when the cache is very large.

Byte Hit Rate — Instead of counting only requests, byte HR (BHR) is concerned with how many bytes are saved. This is the number of bytes satisfied from the cache as a fraction of the total bytes requested by clients. A BHR of 60 percent indicates that 6 bytes will be returned from the cache if a total of 10 bytes is requested.

Note that HR and BHR trade off against each other. In general, keeping more small popular objects in the cache optimizes HR, whereas keeping larger popular objects optimizes BHR. BHR is of particular interest to systems with limited external network bandwidth.

CPU Utilization — Sophisticated replacement policies usually require higher computation overhead. Therefore, besides HR and BHR, CPU utilization is also important to measure the efficiency of a policy. Although it is driven by the cache server's implementation, the big O notation (a mathematical notation used to analyze the complexity of algorithms) can be used to evaluate the complexity of a replacement policy. The complexities of popular policies using the notation are summarized in [5, 6].

This metric is less of a concern in the literature because the CPU is assumed not to be the performance bottleneck. However, in practice, a busy server should not use a policy requiring high computational overhead. If the proxy is overloaded, it will drop incoming requests and cause many connection timeouts, leading to very low effective throughput. Therefore, CPU utilization is of particular interest to busy cache servers or servers with limited processing power.

Latency Reduction — This is the percentage of object download latency that can be reduced. To increase the reduction, a policy should try to cache the object with the highest latency first.

In general, high HR implies high latency reduction because of fewer remote communications. However, it is not always guaranteed. For example, many hits to objects with low latency would result in lower latency reduction than a few hits to those with high latency. Since it is difficult for the cache to measure the download latency, which is affected by many factors outside the cache such as network congestion and server stability, this performance metric is not widely used. It is of particular interest to systems that require low retrieval time experienced by end users.

Many Web cache replacement policies have been proposed and almost all of them were demonstrated to be superior to others in their proposal. However, contradictory results have been reported in the literature. For example, the results in [3] show that the size policy achieves a higher HR than the LRU policy in most situations, whereas the results in [7] show that LRU outperforms size in terms of HR for some cache sizes. Besides, [2] shows that LRU outperforms LFU in terms of HR in most situations under their study, whereas [8] shows that LFU outperforms LRU in most cases.

The above contradictory results show that a policy that performs best in all environments is by no means possible. This is because the performance of a policy has a high dependence on workload characteristics. One workload may make a policy perform well, and another make it perform less well. On the other hand, the above arguments mostly consider HR or BHR only. When choosing a policy to use, we also need to consider other factors such as implementation issue, cache size, processing power requirement, memory consumption, and where the cache server is installed.

Categorization of Policies

The goal of this article is to suggest suitable replacement policies for different environments. However, as there are a multitude of policies, it is unwise to comment on them one by one. In general, a particular category of policies can be suitable in a particular type of environment. We classify replacement policies as:

- Recency-based polices
- Frequency-based polices
- Size-based polices
- Function-based polices
- Randomized polices

Our classification is based on that given in [4] with the modification of adding the size category and removing the recenctness/frequency category. Unlike many of the previous studies that mainly comment on the advantages or disadvantages of different categories, we focus on the design rationale behind the categories and discussing which category works particularly well in which situation (Table 1a).

We find that many policies in the same category provide similar performance. To reduce the complexity when selecting a policy to use, we discuss one representative policy in each category. Through discussion of them, readers should have a basic understanding of the operations and implementation issues of the policies in each category.

Recency-Based Policies — Recency-based polices use recency as the primary decision making factor; most of the policies in this category are LRU variants. The rationale behind this category is that recently accessed objects are likely to be accessed again in the near future.

These policies perform particularly well when Web request streams exhibit high temporal locality. This happens when many clients have a common set of Web objects in which they are inter-

Category	Design rationale	Good for
Recency-based	A recently referenced object will be referenced again in the near future.	When many users are interested in the same Web objects at about the same time.
Frequency-based	Only a small set of objects is popular, and those objects should be cached.	When users tend to access Web sites having objects with quite steady popularities.
Size-based	Removing a larger object can make room for multiple smaller ones.	When users tend to access information-based Web sites.
Function-based	Considering more parameters could achieve a higher hit ratio.	When the system has sufficient processing and memory resources.
Randomized	Complex data structure and high computation overhead are not necessary.	When the system has limited processing and memory resources.

(a)

Category	Available replacement policies	Representative policy
Recency-based	LRU, LRU-threshold, LRU*, LRU-hot, LRU-LSC, SB-LRU, SLRU, HLRU, Pitkow/Recker, EXP1, value-aging, generational replacement	LRU
Size-based	SIZE, LRU min, partitioned caching, PSS, CSS, LRU-SP	LFU-DA
Frequency-based	LFU, LFU-Aging, LFU-DA, Window-LFU, swLFU, Aged-swLFU, α-Aging, HYPER-G	Size
Function-based	GD-Size, GDSF, GD*, PGDS, server-assisted cache replacement, TSP, Bolot/Hoschka, MIX, M-Metric, HYBRID, LNC-R-W3, LRV, LUV, LR, N-gram	GD-Size
Randomized	RAND, HARMONIC, LRU-C, LRU-S, randomized policies using utility functions	Harmonic

(b)

Category	Hit ratio	Byte hit ratio	Complexity
Recency-based	Fair	Fair	Fair
Size-based	Fair	Fair	Fair
Frequency-based	Fair	Fair	Fair
Function-based	Best	Best	Highest
Randomized	Worst	Worst	Lowest

(c)

■ Table 1. *Categories of replacement policy and their a) design rationales and preferable environments; b) available policies; and c) overall performances.*

ested. A proxy serving many clients (a large company or an Internet service provider, ISP) usually exhibits higher temporal locality.

In this category, LRU is the most popular policy. It evicts the least recently referenced object first. This is particularly popular because of its simplicity and fairly good performance in many situations. It is designed on the assumption that a recently referenced document will be referenced again in the near future.

Frequency-Based Policies — Frequency-based polices use object popularity (or frequency count) as the primary factor. The rationale behind is that different Web objects have different popularity values, and only a small set of popular objects account for most of the total requests. Therefore, by trying to keep those objects with high frequency counts in the cache, most requests can be satisfied.

This category of polices is suitable for systems in which the popularity distribution of objects is highly skewed, or in which there are many requests to Web sites having objects with very steady popularity (rarely changing abruptly). Such Web sites include online libraries, distant learning, and online art galleries.

LFU is a simple policy that evicts the least frequently referenced object first. However, it is not recommended because it suffers from the cache pollution problem (i.e., when an object, having accumulated a very high reference count, becomes unpopular, it remains in the cache a long time without being a candidate for removal).

LFU with dynamic aging (LFU-DA) [4], a variant of LFU, avoids the cache pollution problem by using the dynamic aging technique, which adds a constant value to the frequency count of an object when it is accessed, making recently popular objects have larger frequency counts. Since LFU-DA evicts the object with the smallest frequency count, this prevents previously popular objects from polluting the cache. Another advantage of LFU-DA is that it is parameterless. Parameterless policies are preferable as they are usually less complex and easier to manage. Therefore, LFU-DA is a good choice in this category.

Size-Based Policies — Size-based policies use object size as the primary factor, and these usually remove larger objects first. The rationale behind is that most objects in the Web are small in size, and removing a larger object can make room for multiple smaller ones. Therefore, it works well when large objects are less popular. This happens when users tend to access information-based Web sites (e.g., news, weather, and article stores) that contain more text-based than multimedia files.

Size is the representative policy in this category. It evicts the largest object first. Size should be implemented by maintaining a priority queue based on object sizes. As the size of an object is fixed, it requires a constant time for cache hit.

Function-Based Policies — Function-based polices generally associate each object in the cache with a utility value. The value is calculated based on a specific function incorporating different factors such as time, frequency, size, cost, and latency, and different weighting parameters. The object with the smallest value is evicted first. The rationale is that the bottleneck resource in a proxy is network and disk I/O (not the CPU), so it is worthwhile to invest extra CPU cycles to use a more sophisticated replacement policy to achieve a higher HR.

Nevertheless, it is difficult to implement function-based polices because the heavily parameterized function requires a complicated data structure. Besides, the polices require high operational overhead because the data structure has to be updated frequently in some designs. Therefore, this kind of category should be used when the CPU is not limited and the implementation issue is less of a concern.

Greedy dual size) (GD-Size [2] is a function-based policy but its operation is simple. It is reported to perform well in the literature. The policy maintains for each object a characteristic value H_i. The calculation of H_i involves the cost of object i. The cost can be in terms of time or money. The object with smallest H_i should be evicted first. The motivation behind is that the objects with larger fetch costs should stay in the cache longer.

Randomized Polices — Policies with complex data structures motivate the consideration of randomized policies that require no data structures to support eviction decisions. A particularly simple one is RAND [4] that evicts an object drawn randomly from the cache. As this kind of policy requires no state information, both memory and processing power can be saved.

However, policies that use only simple random functions do not provide very good performance. To improve it, some randomized policies apply utility functions to rank the objects in cache, and then randomly pick a victim from a set of samples with the highest utility values. Although, this kind of randomized policy requires more resources, it does not need to maintain and update a data structure (e.g., a priority queue) every time an object is accessed, which function-based policies usually require.

RAND, the simplest randomized policy, evicts an object randomly drawn from the cache. Although this simple policy requires almost no data structure to perform object insertion or eviction, it does not perform well. HARMONIC [4] improves the performance by using a nonuniform probability distribution. First, each object has a cost value (e.g., based on the cache size); then the probability of an object is inversely proportional to its specific cost. In this way the objects with lower costs have higher chances to be evicted. A simple data structure is needed to keep the cost value of objects in the cache.

Performance metric	To be particularly concerned with
HR	Small cache systems.
BHR	Limited external network link systems.
CPU utilization	Busy cache servers or servers with limited processor power.
Latency reduction	Systems requiring low retrieval time experienced by end users.

Table 2. *Performance metrics and their targeted environments.*

Table 1b lists the replacement polices reported in the literature according to their categories. The list is certainly not exhaustive, and some replacement policies could be classified into one or more of the categories. When we assign a policy to a category, the assignment is based on the factor on which the policy mainly relies; it can use further factors to assist its decision making. For example, Log2(SIZE) evicts the largest object first; if two objects have the same size ranking, the least recently accessed will be evicted. It is classified in the size category because it gives first preference to size, then to recency. It is beyond the scope of this article to discuss the operations of the policies listed in the table. Readers interested in policy operations can refer to the original proposals or to [4, 5], which summarize the operations of over 20 policies.

Table 1c shows the overall performance of different categories of policy in terms of different metrics. Among them, the function-based policies provide the best HR and BHR because these policies consider many parameters when making replacement decisions. However, maintaining these parameters in the system causes the highest implementation complexity among all the categories. On the other hand, randomized policies provide the worst HR and BHR because the replacement decisions are made randomly. Nonetheless, this does make them easier to implement.

Note that Table 1c only serves as a general guideline. The HR and BHR performances of the policies are sensitive to workload characteristics, proxy types, system parameters, and available system resources. That is why contradictory results have been reported in the literature, as mentioned earlier. On the other hand, this table only shows the general implementation complexity for each category of policy. For complexity using the big O notation of popular policies, readers should refer to [5, 6]. Besides these metrics, when selecting a policy to use, one should understand a category's design rationale and its targeted environment.

Policy Suggestion

In this section we suggest the appropriate polices to be used in different cases. We classify the cases based on proxy resources and proxy types. We also discuss the selection of policy for commercial and hardware-based proxy products, which are rarely reported in the literature. Table 3 summarizes our suggestions.

Proxy characteristic	Factors concerned	Suggested policy
CPU-bounded	Simple and low operational overhead	Recency-based policies, randomized policies
With small cache	Higher HR	Recency-based policies
With large cache	No particular concern	LRU, etc.
With limited bandwidth	Higher BHR	GD-Size(packet), etc.

(a)

Proxy type	Characteristic	Suggested policy
Proxies in small organizations	Have ampler cache space	LRU, etc.
ISP-level proxy	Higher temporary locality	Recency-based policies
Root-level proxy	Complex traffic characteristic and low temporary locality	Function-based policies using functions with size concerns

(b)

Proxy type	Characteristic	Suggested policy
Commercial-based proxy	Require easier implementation	LRU, site-based LRU, etc.
Hardware-based proxy	Both hardware and software are exclusively designed for Web caching	Sophisticated function-based policies or simple policies

(c)

■ Table 3. *Suggested policies for a) proxies with different resources; b) different types of proxy; and c) commercial proxies.*

Resource-Based Classification

CPU-Bounded — When a proxy is very busy, sophisticated policies that require high computational overhead are not suitable. Those policies would make the busy proxy busier, and even overloaded. When a proxy is overloaded, the performance degrades quickly, and a lot of connection timeouts occur and are reported as errors. Therefore, it is preferable in this case to use simple and efficient replacement policies that do not need to frequently update the data structure or maintain the meta data of cached objects. Recency-based policies are more suitable in this case because they usually can be implemented with a linked list that requires less computation overhead.

Another kind of CPU bounded cache server is a machine with a very low-end processor, such as those used in PDAs or smartphones. These portable devices are becoming popular for accessing the Web. Caching provided by the devices (i.e., client side caching) is useful to reduce traffic sent to the wireless link. In fact, a few years ago, Wireless Application Protocol (WAP) phones already provided a few hundreds of kilobytes cache to store recently accessed WAP pages. In this case a linked list data structure is still complex (or luxury) considering the limited processor power. To solve the problem, randomized-based polices such as HARMONIC or RAND can be used.

With a Small Cache — A cache size is said to be small when it is much smaller than the total size of requested objects seen in the workload. A smaller cache size results in lower hit performance. In the extreme case, zero cache size does not provide any caching function. In order to maximize the advantage of caching (or HR), recency-based replacement policies are preferred to frequency-based in this case because requests to the same object usually exhibit short-term temporal correlations.

With a Large Cache — As the cost of storage is steadily falling, the cache sizes of many caching systems (particularly small and medium ones) are large enough to hold most of the requested objects. As many studies have shown, when the cache size is very large, different replacement policies provide very similar performance. This should be expected because quite often there is a room in the cache for the new objects, and hence a policy does not have to be invoked. In this case some objects in the cache even timeout before they have to be evicted due to cache fullness. In this way the selection of a policy is not significant. Therefore, simple LRU is sufficient in this case.

Podlipnig and Boszormenyi [4] support the above argument by giving the following example. Assuming that arrival rate is 1000 requests/s, average object size is 10 kbytes, 40 percent of data is uncacheable, and BHR is 40 percent, the cache will get 2.05 Mbytes/s. If the cache size is 200Gbytes, at 2.05 Mbytes/s it would take about 28 h to fill the cache; when it is full, the cache contains about 21 million objects. According to the statistics reported in [9], the same object will be requested again after at most 16 million requests. This is 5 million fewer than 21 million. These numbers indicate that even the simple LRU policy would be sufficient for this cache.

Note that in this article we only consider the caching of general Web objects. In some environments such as a video library or software dissemination, the requested objects are very large in size, and a single cache would never be large enough to store most of them. In these cases, special caching techniques such as prefix cache and content delivery network have to be applied.

With Limited Bandwidth — When the external network bandwidth is limited or expensive, it is desired to minimize network traffic. BHR is of particular interest because it concerns how many bytes are saved. In this case size-based polices (e.g., Size) are not suitable because they discriminate against larger popular objects, resulting in low BHR. Some function-based polices that use function to optimize HR at the expense of BHR are not suitable either. For example, GD-Size(1) [2] is designed to maximize HR by setting the transmission cost for all objects to the same constant value, 1. As shown in [2], both Size and GD-Size(1) provide very low BHR compared to other popular polices.

In the literature many function-based policies are reported to provide very good BHR (also HR) because they are usually heavily parameterized. To minimize network traffic resulting from misses, Cao and Irani proposed the GD-Size(packets) policy in [2]. The policy uses a special cost function to assign the cost of an object to 2 + size/536 (i.e., the estimated number of network packets required for a cache miss). Their simulation results show that in most cases GD-Size(packets) achieves better BHR than other policies, including LRV [7], which is sophisticated.

Function-Based Classification

As mentioned before, Web proxy servers are widely deployed in many places throughout the Web. Some are found in small organizations, others in ISPs or the Internet backbone itself. Different types of proxies have different characteristics, and hence require different types of replacement policies. In this section we discuss each type of proxy regarding its system characteristics, workload features, and suitable replacement policy.

Proxies in Small Organizations — Proxies for departments or small organizations serve a smaller set of users than do other types of proxy. In these small environments the typical disk of 200 Gbytes already provides ample cache space to hold most of the requested objects. This implies that proxies of this type essentially have a large cache space.

As observed in previous work, there is no obvious performance difference between replacement polices in a large cache environment. For example, the simulation results reported in [10] show that when the cache size is 20 percent of the total requested size, the policies provided almost the same HR. Besides, [11] shows that when the cache size is 64 Gbytes there is no performance difference among the policies considered in their simulation using a proxy log of requests in a university. For this reason, it is believed that a simple policy such as LRU is sufficient for the proxy in small systems.

On the other hand, previous studies [12] reported that the traffic of local (e.g., department and school) proxies exhibits a higher skew of Zipf distribution than other kinds of proxies. That is, requests are highly concentrated on a smaller set of objects. This implies that a smaller cache space is needed for proxies in small organizations. This also supports the idea that a disk of typical size could cache most of the requested objects.

ISP-Level Proxy — An ISP-level proxy serves a large user population and hence has higher temporal locality because there are objects shared among users. Mahanti [13] studied the workload characteristics of different types of proxies. They observed considerable short-term temporal locality at the ISP-level proxy, showing that approximately 30 percent of the total re-references to objects occur within an interreference time of 1 min.

The presence of short-term locality makes recency-based policies such as LRU more preferable in this case. This claim is supported by [8], which shows that the LRU provides better HR for various cache sizes in an ISP-level proxy.

Root-Level Proxy — Today, many proxies work together to form a cache hierarchy. That is, when a cache miss occurs, a proxy will contact its parent proxy to see if it has the requested object cached. The process continues until the root proxy in the hierarchy has been queried.

In a cache hierarchy root-level proxies receive requests from lower-level proxies located in different areas. As reported in [11], this makes the traffic exhibit lower temporary locality than that of the lower-level proxies. For this reason, replacement policies depending highly on the factor of recency are not appropriate. To increase hit performance, we propose to use function-based polices (e.g., GD-Size) that consider multiple factors.

Our claim is supported by the study of Busari and Williamson [14]. They evaluated the performance of using different replacement policies at different levels in the caching hierarchy. They showed that the effectiveness (HR) of a root-level cache can be much improved by using a policy that takes object size into account (i.e., size-based policies, or function-based policies using functions with size concern). For the scenarios and workloads they studied, GD-Size (function-based) at the root-level cache provides significantly better HR than LRU (recent-based) and LFU-aging (frequency-based).

Commercial and Hardware-Based Proxies

Commercial — Cache vendors launch different kinds of commercial products to cater for different markets. However, it is observed that those products usually differentiate themselves based on user-friendly features, such as plug-and-play installation, multiple protocol support, and easy administration, more than on an advanced replacement policy.

On the other hand, considering the complexity of project development, anything more complicated is undesirable. Therefore, in practice, simple replacement policies should be considered (unless there is a special requirement), and LRU is regarded as the best candidate in this case.

In addition to LRU, site-based LRU [15] is also preferable in this case because it is friendly in performing administrative tasks. Its operation is very similar to that of LRU, and it requires just link lists to implement. It maintains a link list called a *site list*, and each node in the list has a dedicated list called an *object list*. By looking up the lists, it is easy for the proxy software to perform administrative actions such as deleting all cache objects for a given Web site, and queries such as asking which objects are being cached for a given Web site. Commercial products favor the support of these kinds of administrative tasks.

Hardware-Based Proxy — Some vendors design hardware- or appliance-based proxy servers such as Cache Engine by Cisco [16] and Cobalt Qube by Sun Microsystems [17]. Some vendors even have their own proprietary operation systems for proxy servers. Since these kinds of servers are exclusively designed for caching purposes, they are well prepared for the use of sophisticated function-based policies so as to produce a high-performance server. However, as mentioned before, the use of sophisticated policies may complicate the design of a commercial product. Therefore, it all depends on the design goals.

The Future of Web Caching and Replacement Policy

Web caching is playing an important role in the development of the Internet. From its inception, the Internet has not been well prepared for both the rising demand for Web content and the exponential increase in Web traffic. This situation made caching an obvious solution in filling the gap between network capacity and user demand. In 1999 Internet caching grew into a full-fledged market (according to the report by Internet Research Group, a market research firm focusing on

Internet infrastructure technologies). A few years later, IDC's June 2003 Worldwide CDN/Caching Competitive Analysis reported that the caching appliance market would hold steady at 2002 figures ($249 million), the reason being that many corporations (particularly ISPs) have already deployed cache devices in their systems. This indicates that Web caching deployment is not as urgent as before.

Today, the Web caching market is active predominantly in those countries (e.g., Asia-Pacific region) where bandwidth is relatively expensive. According to IDC's market analysis of worldwide secure content and application delivery (including Web caching products) from 2005 to 2009, the Asia-Pacific region will have a compound annual growth rate of 14.3 percent, compared to the projection for North America of only 5.1 percent.

Web caching provides many benefits, but it is only applicable to cacheable objects. An object is said to be cacheable if it can be stored and used to answer a future request. Dynamically generated objects by server-side programs that execute each time a request is made are typically considered uncacheable. Therefore, if the Web were full of uncacheable objects, the utility of Web caching would be very minimal.

Nevertheless, caching is an essential part of the Internet's infrastructure. We believe this will still be true in the future. As pointed out in [18], even as bandwidth costs continue to drop, caching will continue to reap benefits for a number of reasons. First, bandwidth will always have some cost. Second, bandwidth demands continue to increase and, if the price is low enough, demand will always outstrip supply. Furthermore, there will always be variations in bandwidth and latencies in the world, and caching can help iron out these effects.

Web caching has been extensively studied in the last decade. Many issues have been brought up and discussed. The issues include replacement policy, content prefetching, cache consistency, cooperative proxies, caching protocol, and cache deployment. Please refer to [19] for the general discussions of them. There are even dozens of caching products available on the market. Although Web caching in its general form seems to be a solved topic, there is still a gap in the literature regarding special systems, such as multimedia (with many large objects), wireless (with higher latency and error rate), and mobile (with mobility and smaller cache) systems.

Similarly, the research of Web caching replacement has been active in the last decade. We believe there are enough sufficient good replacement policies for general Web proxy, and new proposals only provide tiny incremental improvements. However, there are particular environments requiring proposals of new specialized replacement policies to improve performance. One of the environments is an energy-critical system (i.e., energy consumption is a concern). For example, Zhu *et al.* [20] proposed a new power-aware cache replacement policy to reduce energy consumption. By trying to keep the disk block with larger energy penalties in the cache, the policy saves 16 percent more disk energy than the conventional LRU in their trace-driven simulations. Another environment where replacement policies can help to improve performance is wireless. For example, Katsaros and Manolopoulos [21] proposed a new self-tunable cache replacement policy to reduce latency and conserve network resources in broadcast mobile wireless environments. The policy slices the cache space and employs intelligent methods for selecting the replacement victim. It provides much better performance than conventional LRU in terms of average stretch (the ratio of an object's access latency to its service time). On the other hand, with the recent rapid growth in mobile Internet, the use of transcoding proxy servers [22] will become more and more popular. Transcoding is the process of converting an object from one form to another (e.g., from a true color to a grayscale image). Yeung *et al.* [22] proposed a new replacement policy that takes the transcoding time into account when evicting an object. Simulation results show that it outperforms conventional LRU by reducing the average transcoding time by about 40 percent. In the future, efforts should be given to propose new policies for environments with special needs.

Conclusion

In this article we have made a number of insightful observations and contributions to the literature on Web cache replacement. First, we establish a position on the diminishing need to devise new replacement policies for traditional Web caching environments. Second, we summarize common observations spread over much previous work. Most important, we offer a comprehensive collection of suggestions about which policy could be most effective in which environment. The suggestions are based on the resource (e.g., cache size, bandwidth, and processing power) and functional characteristics (e.g., ISP- and root-level) of Web proxy caches.

References

[1] A. Luotonen and K. Altis, "World-Wide Web Proxies," *Computer Nets. and ISDN Sys.*, vol. 27, iss. 2, 1994, pp. 147–54.

[2] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms," *Proc. USENIX Symp. Internet Tech. and Sys.*, Dec 1997, pp. 193–206.

[3] S. Williams *et al.*, "Removal Policies in Network Caches for World-Wide Web documents," *Proc. ACM SIGCOMM*, CA., Aug. 1996, pp. 293–305.

[4] S. Podlipnig and L. Boszormenyi, "A Survey of Web Cache Replacement Strategies," *ACM Comp. Surveys*, vol. 35, no. 4, Dec. 2003, pp. 374–98

[5] A. Balamash and M. Krunz, "An Overview of Web Caching Replacement Algorithms," *IEEE Commun. Surveys & Tutorials*, 2nd qtr. 2004, vol. 6, no. 2.

[6] B. Hyokyung *et al.*, "Efficient Replacement of Nonuniform Objects in Web Caches," *Comp.*, vol. 35, no. 6, June 2002, pp. 65–73.

[7] L. Rizzo and L. Vicisano, "Replacement Policies for a Proxy Cache," *IEEE Trans. Net.*, vol. 8 2, Apr. 2000, pp. 158–70.

[8] L. Breslau *et al.*, "Web caching and Zipf-like Distributions: Evidence and Implications," *Proc. INFOCOM*, vol. 1, 1999, pp. 126–34.

[9] M. F. Arlitt, R. J. Friedrich, and T. Y. Jin, "Workload Characterization of a Web Proxy in a Cable Modem Environment," *ACM SIGMETRICS Perf. Eval. Rev.*, vol. 27, no. 2, 1999, pp. 25–36.

[10] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms," *Proc. USENIX Symp. Internet Tech. and Sys.*, Dec 1997, pp. 193–206.

[11] A. Mahanti, D. Eager, and C. Williamson, "Temporal Locality and its Impact on Web Proxy Cache Performance," *Perf. Evaluation*, vol. 42, no. 2–3, October 2000, pp. 187–203.

[12] A. Mahanti, C. Williamson, and D. Eager, "Traffic Analysis of a Web Proxy Caching Hierarchy," *IEEE Network*, vol. 14, no. 3, May-June 2000, pp. 16–23.

[13] A. Mahanti, "Web Proxy Workload Characterization and Modeling," M.Sc. thesis, Dept. of Comp. Sci., Univ. of Saskatchewan, Sept. 1999.

[14] M. Busari and C. Williamson, "On the Sensitivity of Web Proxy Cache Performance to Workload Characteristics," *Proc. IEEE INFOCOM*, vol 3, 2001, pp. 1225–34.

[15] K. Y. Wong and K. H. Yeung, "Site-Based Approach in Web Caching Design," *IEEE Internet Comp.*, vol. 5, no. 5, Sept./Oct., 2001, pp. 28–34.

[16] Cache Engine. http://www.cisco.com/

[17] Cobalt Qube. http://www.cobaltnet.com/

[18] B. D. Davison, "A Web Caching Primer," *IEEE Internet Comp.*, vol. 5, no. 4, July-Aug. 2001, pp. 38–45.

[19] M. Rabinovich and O. Spatscheck, *Web Caching and Replication*, Addison-Wesley, 2002.

[20] Q. Zhu *et al.*, "Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management," *Proc. 10th Int'l. Symp. High Perf. Comp. Architecture*, Spain, Feb. 14–18, 2004.

[21] D. Katsaros and Y. Manolopoulos, "Web Caching in Broadcast Mobile Wireless Environments," *IEEE Internet Comp.*, vol. 8, no. 3, May–June 2004, pp. 37–44.

[22] K. H. Yeung, C. C. Wong, and K. Y. Wong, "A Cache Replacement Policy for Transcoding Proxy," *IEICE Trans. Commun.*, vol. E87-B, no. 1, Jan. 2004, pp. 209–11.

Biography

KIN-YEUNG WONG (kywong@ipm.edu.mo) received his B.Sc. and Ph.D. degrees, both in information technology, from the City University of Hong Kong. He is currently an associate professor at Macao Polytechnic Institute. He is active in research activities, and has served as a reviewer and technical program committee member in various journals and conferences. His research interests include Internet caching systems, wireless communications, and network infrastructure security.