Concept of Cache
in web proxies

Chan Kit Wai and Somasundaram Meiyappan

1. Introduction

Caching is an effective performance enhancing technique that has been used in computer systems for decades. However, proxy caching differs substantially from the traditional ones used in processors. Two main features of the World Wide Web applications that differ from the traditional caching are (i) non-uniformity of the object sizes and (ii) non-uniformity of the cost of cache misses (as opposed to the traditional caching where all cache blocks have the same size and require the same amount of time to be retrieved in case of cache misses). The traditional metrics for measuring caching efficiency has been hit-ratio (HR), which is defined as the number of requests satisfied by the cache divided by the total number of requests. Obviously, HR is not an appropriate metric to measure performance of proxy caches, because of the non-uniformity of object sizes and non-uniformity cost of misses. Byte Hit Ratio (BHR) is defined as the number of bytes found in the cache divided by the total number of bytes requested within the observation period. The BHR takes into consideration the non-uniformity of the object sizes, but it fails to consider the non-uniform cost of misses.

Another phenomenon is that users' interests overlap in time, which indicates that part of the locality observed by the proxy comes from the fact that the proxy sees a merged stream of accesses from many independent users, who share a certain amount of common interests [1].

2. Basic Unit for storage of proxy cache

In the field of computer architecture, the basic unit of storage in a cache is the 'cache line' which is a group of bytes. For proxy caches, we could either have objects as the basic unit of storage or packets, which make up an object, as the basic unit of storage. In this section, we will discuss reasons on why bytes cannot be the answer.

The input to a cache in computer architecture is the address of the memory location to be accessed. But in the World Wide Web, proxy caches receive request to objects as the input. Based on the protocol used, all objects are broken into packets of data. But if packet is used as the basic unit of storage, then it is possible that some packets of the requested object is in the cache whereas the other packets might not be in the proxy cache. While such a system could help objects like html files which can partly be reconstructed from the initial packets, it cannot be used for objects like GIF and JPEG. It would still not reduce the latency of obtaining an object as the proxy has to go to the destination server to get the remaining packets.

These reasons point to the fact that for effectively reducing latencies diving browsing, the entire object needs to be cached. But for very large objects caching packets could help reduce the

overall bandwidth usage in the destination server. So, overall for caching web-pages, which are normally composed with html, GIF and JPEG, object seems to be the logical choice for the basic storage unit in a proxy cache.

3. Reference localities observed in web object retrieval

Proxy cache works on the principle of localities of reference. The different types of localities can be explored to exploit the performance of proxy caching. The key objective in this section is to explore which kinds of localities that are relevant in proxy cache and those that can be employed as file replacement algorithm for good effectiveness of proxy cache.

One of the most commonly used proxies (LRU) exploits the principle of temporal localities. Each user's accesses tend to exhibit temporal locality. An average user often tends to re-access recently read documents, and re-access documents that are read on a daily basis. Another scenario that exploits temporal localities is that, many independent user accesses to a common site over a period of time. This is indeed true for sites that are frequently access by a group of user that share a common interest.

Spatial locality is another kind of locality observed for web object retrieval. This property is very important in microprocessor cache design and based on this property, programs tend to execute instructions in sequence and are normally in nearby memory locations. As a result, fetching of instruction can be predicted rather accurately. In contrast, the requests of web objects are definitely not in sequence. It is neither in sequence with respect to memory location as well as files location. The next request can be anywhere on an object within the site or another site. Also, spatial localities in processor caches arise from the fact that different variables or part of the same variable might be in the cache line. But in proxy caches, the storage unit is an object. So, spatial locality cannot be noticed. The concept that comes closest to spatial object is that when an object has a hit, the entire object is served from the proxy cache.

4. Observation on the proxy cache hit ratio.

From the results obtain from the simulation of the bigger trace cache that was provided, we understand that the cache hit ratio depends on a number of factors. The hit rate is dependent on the size of the cache, the caching algorithm and the total client population that the proxy server is supporting.

First, the cache size plays an important role. An ideal cache size would be that of an infinite cache. It would enable all web objects to be cached to the proxy without any replacement policy. In practice, an optimal cache size is necessary to trade off between cost and performance. From the simulation results obtained from figure 1, the size of the cache can be set according with respect to different caching algorithm.

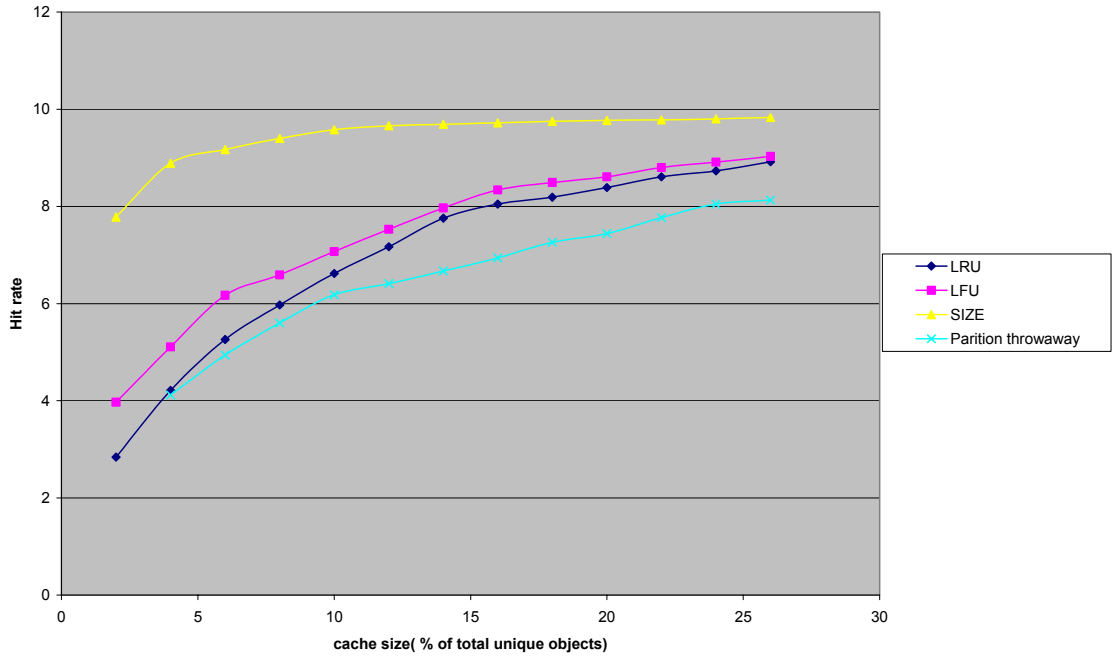


Figure1: Cache size vs. Hit rate (simulated with 10000 requests)

Secondly, a good caching algorithm can help to improve the hit rate. For instance, the size caching algorithm can achieve comparative higher hit rate with respect to other algorithms. However, it is important to note that the simulated result is carried out base on only 10000 requests. That means, the performance of different caching algorithm may have their peak performance (hit ratio) at different cache size as well as client population.

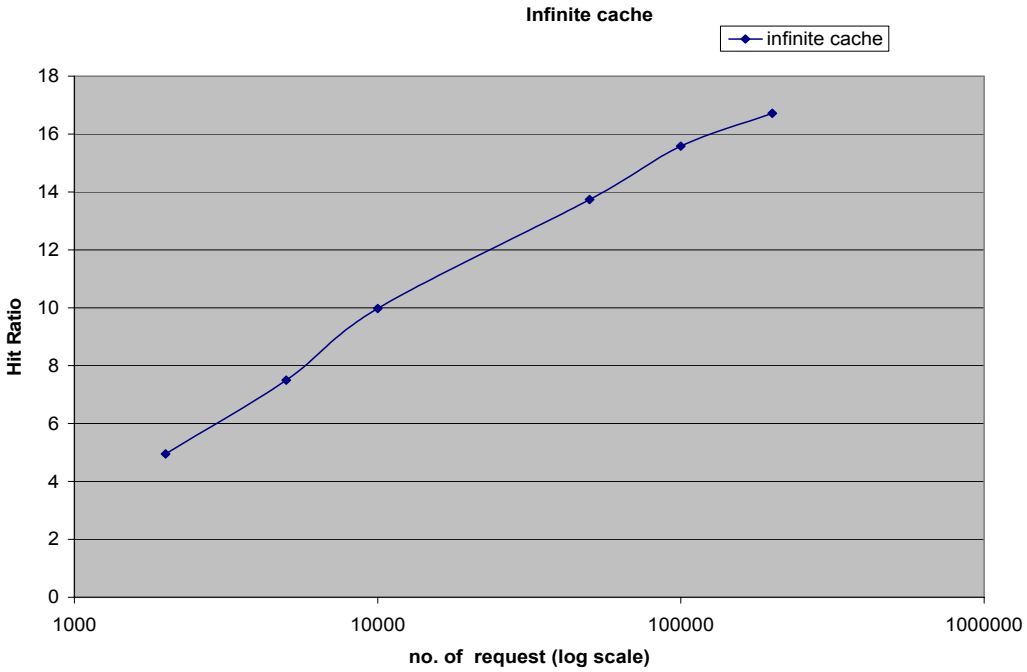


Figure 2 : No. of request vs. Hit rate (infinite cache size)

An experiment is also carried out to determine the effects of client's request on the hit rate. In the experiment the cache size of infinity is taken to be the reference as to simulate the best algorithm. At the cache size of infinity, the hit ratio is independent of any caching algorithm. Without practical considerations, infinite cache size would mean the best caching algorithm that one can easily employ.

The simulation results obtained in figure 2 shows the relationship between the hit rate and the number of request. From another point of view, the number of request is also related to the client population that the proxy cache is supporting. Hence, it can be concluded that the hit rate increases linearly with the logarithmic of the client population. To put it simply, the proxy cache performance will increase predictably linear if the population of it's client increases in an exponential rate. It is interesting to note such a relationship; as to whether if this relationship will still hold to achieve a hit rate of 100%.

It is also interesting to note when simulating older traces (1996) that were largely picked from local proxies in universities and companies, the hit ratio is observed to be very high-48% to 60%. This point to the fact that in business corporations and universities there is a high temporal locality than in a more diverse proxy that is present with the ISP or in the internet backbone. So, the location and the purpose of the proxy might also need to be considered when deciding on the algorithm to use in proxy cache.

Collectively, an average proxy cache which performs sub-optimally is typically found to have hit rates below 55% [1]. In comparison, the latest reports on the trace cache for Pentium4 processor will have a 95-98% average hit-rate for most operations [2].

The fact that processor cache achieves a close to perfect performance is because programs inhibit a high level temporal and spatial locality. Most programs exhibit "temporal locality", which means that once a memory location that has been referenced, it is likely to be accessed again soon. From statistics, most of the programs are executed in a loop. Hence the instructions inside loops are fetched and executed many times. On the other hand, the proxy cache does not exhibit as much temporal behaviour.

Another property of program behaviour which is important to processor cache design is "spatial locality". This increases the efficiency of cache operations by increasing the data transfer rate, or "bandwidth", of the memory/cache interface. Additionally, the properties of spatial locality allow the processor to predict and prefetch the next instruction and data.

On the other hand, web object retrieval does not exhibit the properties of spatial localities. Each request can be very independent from its next request. However, a similar approach can be applied to proxy cache to "prefetch" web objects that are of neighbours to the current requested object. The details of prefetching will be discussed in the later section.

5. Performance parameter for web proxy cache

Most of the performance studies on proxies to-date concentrated on hit ratios [4]. The hit ratio that is commonly used as the primary metric for the proxy cache may not actually reflect it's actual performance. There are other important metrics, such as byte hit rate, bandwidth savings and latency reduction. The hit rate is computed base on only the total object request and the total

request hit. It has not taken into the account of the cost and the size of the object. In the web context, replacing a more recently used but large file with smaller but most frequented object can yield a higher hit ratio than replacing a less recently used but small file. In addition, a proxy with a very large disk cache can provide a high hit rate. However if the server suffers from poor disk management, it will be noticeably increase the average latency. Thus, the algorithm should combines locality, size and cost considerations in a simple online way that reduces the total cost. Hence it is important to have a metric that includes both the cost and size of the object as well.

If hit rate may not be the best metric to measure performance, we may consider other metrics or a hybrid of all metrics. In particular, from a client's point of view, they are more concerned with the latency rather than the hit rate. However, employing caching algorithms that are targeted to reduce latencies does not perform well. Instead, it is found that algorithms which aims to maximizing the hit rate also reduced latency more effectively than latency-conscious algorithms. Detail analysis of the traces reported in [5] shows that a document that was considered cheap (taking less time to download) may actually turn to be expensive at the next miss. Hence, even if the latency metric is used for performance evaluation, the variance among the latency made it difficult to implement latency-conscious algorithm.

Based on the metrics mentioned, the proxy cache is to decide whether to cache a large number of small objects (which could potentially increase the hit rate) or to cache a few large objects (possibly to increase the byte hit rate). The SIZE algorithm which only uses the hit ratio is simulated as shown in figure1. It's objective is to replace the most recent object with the largest object in cache. Hence, it aims to maximize hit rate by caching a larger number of small objects.

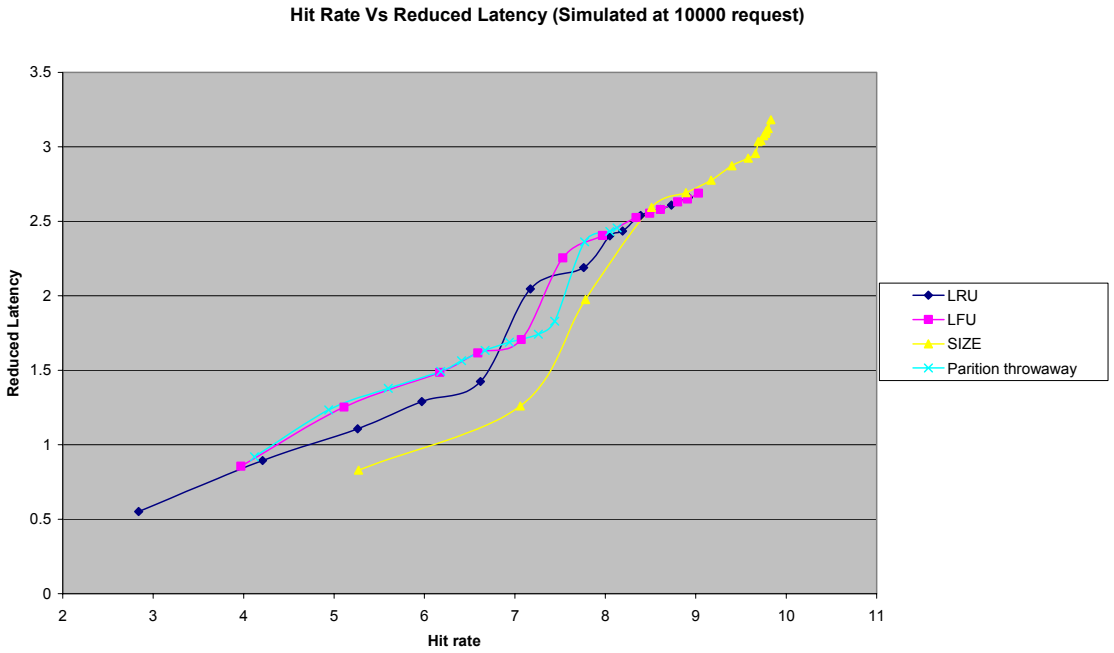
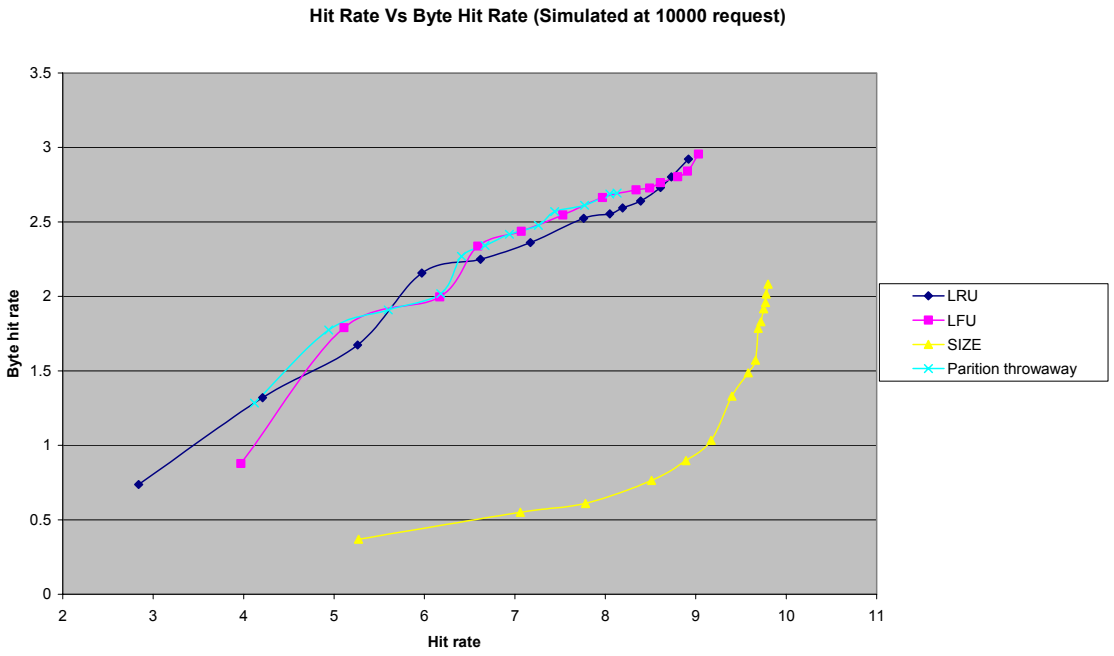


Figure 3: Hit rate Vs Reduced Latency.

Comparatively, experimental results obtained in the figure 3 shows the relationship between the hit rate and reduced latency for different algorithms. The simulation based on a fixed 10000 request from the trace file and statistic for hit ratio and reduced latency is generated using different cache size.

Interesting observations can be made from the SIZE algorithm's hit ratio and reduced latency. For a hit ratio below 8.51, the SIZE algorithm experience a lower reduced latency compared to others. In addtional, the SIZE algorithm also experience a much lower performance measures in byte hit rate, reduced latency as well as reduced packets. However, for hit rate above 8.51, the SIZE algorithm achieved very good results in the hit rate as well as reduced latency.



Other studies have been carried out on the evaluation of caching algorithm on byte hit rate. This metric takes into the consideration of the request byte hits instead of number of request object hits. It provides a finer granularity for the evaluation proxy cache. If the cache hit happens to occurred on one of the large object it has a noticeable effect on the byte hit rate for that run. Hence, this metric have a direct relationship with respect to the bandwidth saving that can be seen from the network.

Other interesting observation is from the relationship of hit rate and byte hit rate. While most of the algorithm have similar characteristic, the SIZE algorithm exhibit interesting results. For a corresponding hit rate, this algorithm produces a comparatively low byte hit rate. It can be due to the effect of storing many small objects and evicting large objects. As a result, other algorithm will have more impact on the byte hit ratio when there is a cache hit on a large object. On other hand, the SIZE algorithm achieve high hit rate on many small object but low byte hit rate due to the small object size.

From the experimental results and observations, it can be seen that none of the single performance measure take into consideration of all aspects. Instead, the evaluation of the proxy performance should employ a hybrid approach. All measures are taken into account and each of the measure should carry a certain weight. The sum of all; shall then be the measure of the cache performance. As a result, this scheme can prevent policies in pushing the performance measure in one area but neglect others, hence achieving an overall balanced approach.

6. Prefetching in web cache

Prefetching is employed for fetching objects from the web server or from the proxy cache in advance so that the user experiences virtually no or very low latency if the user requests these prefetched objects. Sometimes, these prefetched objects might not viewed by the user at all. This is the traditional misprediction scenario.

6.1. Suggestion to prefetching in web cache

Although web objects may always not be requested in sequence, we can derive a relationship between different requests. The idea is actually borrowed from computer architecture design of a branching unit. The PowerPC 750 uses instruction encoding to predict whether the branch is likely to be taken and begins fetching and executing along its predict path. As such, the complier provides hints to the architecture to predict a branch taken or not taken. A suggestion for the prediction of new request for prefetching in proxy is described in the following section.

The idea is that, the server will provide some hints to the client as to what are the objects to be prefetch next. To perform this, the server must analysis the client's access pattern of objects on a particular server. For instance, the server can records the relationship of the current request to it's previous requests. A dynamic analysis of the trace must be performed to collect the relationship between requests. A link-list can be formed to record a list of descendant requests. Hence, with a rich knowledge of user requests for a particular server, it is able to provide a kind of prediction that resembles the sequential execution of instructions within a microprocessor.

The key idea is to provide the client some hint so that it will be able to improve the accuracy of prefetching web objects. For instance, when a client sends a new request to a server, it shall receive the web object as well as the descendant request link-list along with it. The client upon receiving the list will prefetch the relevant web objects. If the prediction is correct, it will be a cache hit and hence help to reduce the download time. In the case of a misprediction, the request will be sent to the server as per normal.

This scheme which helps to prefetch web objects in advance may help to reduce the download time, but can also increase the load on the network if misprediction is large. In the case of a misprediction, the web objects in the cache are not flush away; instead they can be kept for a cache hit in the near future.

Another approach that the proxy can make use of is based on the timely and periodic access patterns of the users of the proxy. For example, every morning, the users of the proxy might probably visit news websites. So, the proxy may try to prefetch objects from these web sites when the network traffic is low.

6.2. Comparison with prefetching in processors

Processors employ two kinds of prefetching. The first type is through an explicit instruction from the program being executed. The second type is based on a dynamic analysis of "program counter" status and memory access patterns. In the second approach, the processor fetches the

data by itself without explicit instructions from the program. As far as the proxy cache is concerned, if the browser tries to prefetch the objects then this would fall under the first approach. But if the proxy cache makes its own decisions on prefetching, then this would fall Into the second approach.

However, there are several drawbacks. Because frequently retrieved resources are only a small fraction of all retrievals, it is difficult to guess the client's next request by any statistical method. User interests tend to change rapidly. To get a high hit rate, this scheme requires a long observation time. It is difficult to catch new trends immediately. Although prefetching improves the hit rate of frequently retrieved resources, its overall reduction of WWW latency is small.

The impact of prefetching not alone affects the performance of the latency, but it does have negative effect on the network. For instance, the proxy can prefetch as many objects as possible, targeting to achieve a high hit ratio. In such scenario, the proxy will achieve a high hit ratio and the client will experience a low latency. However, excessive prefetching can overload the network and causes undesirable wastage of bandwidth.

Finally, objects that have been cached using prefetch might not be in cache for long enough till the user in fact makes the request for these objects. Such cases could increase the server-proxy bandwidth.

7. Cache partitioning

As noted in the first section m the basic storage unit in a proxy cache, different types of web objects show different characteristics. HTML files are normally smaller in size, a few kilobytes and GIFs and JPEGs start at a few tens of kilobytes to even one or two Megabytes. Though these are the major types of objects used in the web, there are other kinds of documents and objects like mp3 files, office documents, portable document format (pdf) files, compressed archives (.zip, .tgz) and executables that are frequently downloaded.

Since most web surfers see only web pages with HTML, GIF and JPEG, it definitely makes more sense to cache these objects. But the other kinds of documents' might need to be cached as well. For example, the proxy server in a university might see more requests for postscript and pdf files just like proxies of a research institution. And these files could even be megabytes in size. Here is the motivation to partition the cache into multiple partitions that cache only certain kind of objects. Some of the factors or criterions can be

- The size of the object: Objects larger in size can be placed in a separate partition
- Frequency of reference: One of the algorithms implemented in the simulator is such that frequently referred objects are moved to a special VIP cache partition.
- The type of the object: One partition for standard web object types, another for archives and executables and the third one for other kinds of objects like pdf, shockwave flash documents, etc…
- Dynamism of the content: Statistics shows that more than 80% of the world's web pages do not change over several months. Only few web objects change frequently. So, partitions can be created to separately cache these kinds of data. Implementation of such an algorithm might involve a study or research.
- Access latency of the destination server: Partitions can be based on the time it takes to retrieve an object from its destination server per byte. The reason why we add the 'per byte' tag is so that larger objects do not get any undue advantage because of their size.

Partitioning caches can also be formed by combining more than one criterion listed above.

8. Multi-level caching system

With the exponential increase in web traffic, the standalone proxy cache and reverse cache on the server side will not be able to support the traffic in the near future. One of the early researches into multi-level caching is carried out by NLANR, the National Laboratory for Applied Network Research. They aim to develop and deploy a prototype of a global web caching hierarchy, with root caches located at each node of the NSF's high speed backbone service. The result of multi-level web caching has experienced a steady increase in usage since its inception in December 1995 [6]. Additionally, the CRISP cache is another other cooperative Internet caches which consists of a group of cooperating caching servers sharing a central directory of cached objects.

8.1. Co-operative caching

In co-operative caching mechanisms, a group of caches (read as proxy servers) work together by collectively pooling their memory resources to provide a larger proxy cache. These co-operating caches can also be centrally managed by a server. To probe the cooperative cache, the proxy forwards the requested URL to a mapping server.

The use of a central mapping service distinguishes the CRISP cache from other cooperative Internet caches. In other cache systems, each proxy probes the cache by multicasting queries to all of its peers, to avoid relying on any single peer or a mapping server. If the request generates multiple hits, the requester can select its preferred site based on time to respond. Unfortunately, multicasting increases network traffic and forces all caches to respond to each request in most cases. In effect, every caching server incurs the load of cache probes that are limited to the mapping server in CRISP.

Looking closely, this technique is very similar to the case in which data requested by a processor is returned from another processor’s cache instead of the main memory in shared memory multiprocessor systems.

8.2. Hierarchical caching

If co-operative caching borrows ideas from the multiprocessor systems, we could have another multi-level caching system built on the traditional concept behind L1/L2/L3 caches used in processors.

A typical scenario would be use of a proxy server in a corporate environment which in turn connects to the ISP’s proxy cache. If we consider the browser cache as Level 1, then the corporate cache would be Level 2 and the ISP’s proxy cache would be Level 3 cache. Just as in the processors, the browser cache (L1) services only a Single client and would hence be smaller in size than the corporate proxy cache (L2) which services web clients from a whole company. And the ISP’s cache would serve a wide variety of web clients from a subscriber to a corporation.

So, as the level of the cache increases, the Size of the cache would also increase and would be able le cache objects retrieved by a wide variety of access patterns.

Multi-level cache aims to spread the cached objects over several associated server. The top level of the multi-level cache hierarchy holds the most commonly accessed objects while the lowest level caches the least frequent ones. A cache may make use of neighbour caches. A neighbour cache can be either a parent or a sibling that receives ICP (Internet Cache Protocol) queries from its neighbours. The concept of such a hierarchical system is to distribute the objects into different servers.

It is observed that both processor's hierarchy cache and web multi-level cache objective is to reduce access latency. While, the processor main concerns are of the load and store latencies, the web caching aims to reduce the overall network load. However, both of these caching schemes shared some similarities. In terms of physical placement of the caches, both of the highest level cache is located near the CPU and the internet client. In addition, both caches aim to achieve the highest hit rate in the top level cache. This is especially important, since hit rate has been often used as a parameter for the cache performance which is closely related to the reduction in latencies.

Despite of the similarities between multi-level cache for processor and the web, they are fundamentally different in several aspects. Basically, the web cache hierarchy architecture is different from that of the processor's cache. Multilevel caching works almost the same as caching with single-cache servers. However, if there is a cache miss at one server level, the request is propagated up to the next higher level to see if that cache contains the object. The web cache hierarchy may consist of several parents and siblings caches. Consequently, when a cache miss occurs, there are several paths to fetch the objects from. In contrast, the processor cache only propagates it's request to a single higher level cache.

As mentioned above, the next higher level may consist of several parent caches. As a result, it is necessary to decide which path the request is to be propagated. The selected path can actually determine access time of the object. In addition, the consideration for the topology of servers is to be taken into account; else one may end up generating more traffic than without web caching. Hence, the web cache architecture poses a challenging area in defining policies for such propagations.

9. Conclusion

In the sections above, we have noted that there are many similarities between processor caches and web proxy caches. This is not coincidental however. Decades of research in computer architecture feeds the research on web proxy caches. In the years to come, this trend would probably continue and good knowledge of computer architecture, large scale networks and an analysis of our access patterns in the web could lead to better proxy cache configurations.

10 . References

1. Greedy Dual-Size algorithm
   http://db.uwaterloo.ca/~nzhong/research/courses/cs740_project/p740/node10.html

2. Intel Pentium 4 Willamette Preview
   http://sysopt.earthweb.com/articles/p4/index3.html

3. A Simple Yet Robust Caching Algorithm
   Based on Dynamic Access Patterns,
   http://archive.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/pitkow/caching.html

4. Performance Effects of the Proxy , http://www.cs.wisc.edu/~cao/WISP98/html-versions/anja/proxim_wisp/node10.html

5. Cost-aware WWW proxy Caching Algorithm, Pei Cao and Sandy Irani

6. Evolution of the NLANR Cache Hierarchy:Global Configuration Challenges, Duane Wessels and k claffy