Modelling and Analysis of Push Caching

R. G. DE SILVA
School of Information Systems, Technology & Management
University of New South Wales
Sydney 2052
AUSTRALIA

Abstract: - In e-commerce applications, information contained in the objects delivered by the server to the clients can rapidly change with time. Therefore, the challenge is not only to locate a cache that contains the required object but also to make sure that the contents of the object are consistent. In this paper, we present a new mathematical model for cashing and analyse the push cashing. We introduce two quantities called push radius and pull radius and show that to decrease the staleness of an object, we have to push that object to a cache within the pull radius by a special means of communication.

Key-Words: **-** Web caching, server pushing, dynamic objects, staleness, push radius, pull radius

1 Introduction

The rapid growth of the Internet users and the popularity of e-commerce increase the network and server loads and the latency of retrieving the requested objects. To reduce these problems web caching was introduced. Caching stores frequently accessed objects in a location closer to the client. If the same object is requested later, and the cached copy is still valid, it is delivered from the cache instead of the origin server. Caching can significantly reduce latencies and network bandwidth. There are two main types of web caches, namely browser caches and proxy caches. Browser caches are associated with the client applications whereas proxy caches are located at intermediate places between the client and the origin server.

It is possible to arrange caching proxies in hierarchical or distributed fashion. Usually, an object is delivered to a cache from the origin server only after a client has made a request for that object. This is known as the client pulling. Harvest or Squid cache [1] belongs to this category. Recently, distributed caching (presented in [2]) that uses server pulling to distribute objects to the caches has been developed. It is claimed that the server pulling has the benefit of reducing the latency of object retrieval as it places the objects closer to the clients. Moreover, it is believed that clients receive the most recent version of the object thereby increasing consistency.

In this paper, we develop a mathematical model for the client server communication via a set of proxy caches and systematically show what real advantages can be achieved by using server pushing in contrast to client pulling. In Section 2, we discuss different proxy caching methods and investigate whether they use a pull technique or a push technique. In Section 3, we construct our mathematical model and use it in Section 4, to analyse the two cases of pushing and pulling. In light of this analysis, we discuss the advantages and disadvantages of the two techniques. The paper is concluded in Section 5.

2 Background

Traditionally, caches have been implemented in a hierarchy as in Harvest or Squid [1] cache. Here caches have parent and child relationships with each other, with the parent being one level up in the hierarchy [1]. A cache checks with its peers whether they have the requested object. If a neighbour returns a 'hit', the object is retrieved from the neighbour. If a request returns a 'miss', the cache requests the object from its parent. The parent cache resolves the request recursively until the object is found. Although a hierarchical cache system shares data among clients to improve hit rates, it increases load on the caches, number of levels in the hierarchy and the distance between the client and the cache [2]. As the requests pass from one cache to another in the hierarchy, the hierarchical caching system essentially belongs to client pulling.

In contrast, a distributed system of caches maximises hit rates, improves hit and miss times. Furthermore, it complies with basic cache design principles such as minimising the number of hops to locate data, ability to share data, scalability, and caching data closer to the client [2]. A distributed system of caches comprises of a group of caches with no hierarchy, allowing complex inter-cache relationships [2]. This system introduces the method of location hints. Location hints allow caches to locate the presence of objects in nearby caches without much data. Caches can then make a cache-to-cache transfer rather than through the hierarchy avoiding store and forward delays. Misses are detected through the hints, so the requests can be sent directly to the origin server [2].

This technique also pushes data closer to clients by guessing the future accesses of clients, thus avoiding the compulsory miss that occurs when an object that has not been requested previously or that is expired is requested. Push caching method known as 'push-on-update' pushes the updated object to a list of caches that previously cached the old version whereas the method known as 'push-shared dynamically' builds a distribution tree for popular objects. Distributed caching has been found to provide speedups of 1.27 to 2.43 compared to the traditional cache hierarchy [2]. Distributed reverse proxy server caching allows many client server relationships and helps overcome delays due to congestion or routing over large distances.

Object search in a cache hierarchy can be improved by using cache digests. Cache Digests allow proxies to make information about their cache content available to peers in a compact format. A peer uses digests to identify co-operating caches that are likely to have a given web object. This is implemented in Summary Cache [3]. Super Proxy Cache [4] builds a simple hash code of the object and directs the request to a cache server that stores the objects of that particular hash code. Both these techniques are very similar to cache hints.

There are several protocols used for inter-cache communication. Internet Caching Protocol (ICP) [5] is used to resolve the objects in hierarchical caching. In ICP, the objects are identified simply by their URLs and as a result, the contents of the object may be inconsistent. That is, a retrieved object may be stale. An ICP query/reply also adds an additional delay to the transaction. Efficiency of ICP can be improved by forcing to resolve non-cacheable and local URLs directly from the object's origin server or by sending requests directly to parent caches bound to that URL's domain [1].

Unlike ICP, Hyper Text Caching Protocol (HTCP) permits caches to mark some objects as uncacheable [6]. For example, the object will not be cached if it is authenticated or secure. HTCP supports 'cache push' by informing its neighbours about significant events (e.g. the expiry of a cached copy) without receiving a request for the object concerned. Another protocol known as Cache Array Routing Protocol (CARP) is essentially free of protocol overhead [7]. It distributes load across an array of proxy servers. CARP uses program code specific to the CARP-compliant proxy servers that causes the client to select a different server for each URL it requests, based on a hash function that takes into account the URL and the capabilities and configuration of the proxy servers. If the proxy servers are configured with a compatible algorithm, they can also infer from a given URL which other servers can deliver the object.

With Web Cache Coordination Protocol (WCCP), the clients send web requests directly to the origin server. Cisco IOS routers intelligently intercept HTTP requests and transparently redirect them to a Cisco Cache Engine [8].

There are other techniques that compliment caching. Replication uses multicast to push data to 'mirror' origin servers whenever the content changes. As multicasting is used for replication, additional protocols are required to ensure recovery of lost packets.

Consistency mechanisms were designed to ensure that cached copies of data are up to date. Several cache consistency mechanisms are currently used on the Internet such as time-to-live (TTL) fields, client polling, and invalidation protocols. The 'Expires HTTP header' or TTL field tells all caches how long the object is fresh for [9]. After that time, caches will always contact the origin server. Client polling is a technique where clients regularly check with the server to determine whether cached objects are still valid. When a cache has an object containing a *Last-Modified* header, it can ask the server if the object has changed with an *If-Modified-Since* request. The *If-Modified-Since* request header field indicates that the server should only return the requested information if the contents have been changed since the specified date. Most web proxies are using this field today. Invalidation protocols are required when weak consistency is not sufficient. They rely on the server keeping track of cached data; each time an object changes, the server notifies caches by pushing the new version. Problems of invalidation protocols are that they are often expensive and they need the servers to keep track of the cached objects introducing scalability problems. Moreover, they need modifications to the server and make the server to attempt repeatedly to contact unavailable clients.

All of the methods and protocols mentioned above claim their advantages and disadvantages based on simulations or real world implementations. To the best of our knowledge, there is no mathematical model developed or the predictions of performance are based on such a model. We recognise that an approach based on mathematical reasoning would lead to gaining better insight and the development of better caching techniques.

3 Mathematical Model for Push Caching

Our approach in constructing the model is to first build a simple model and after gaining some insight, to develop a more complete model. Consider that we have an infinite number of proxy cache servers located in between the origin server and the client as shown in Fig 1. These severs are connected serially by links that are infinitesimally short. The origin server generates continuously a different version of the same object and sends it to the first proxy cache. The first proxy sends it to the second and so on until it reaches the client. Assume that the time for caching a version of the object in a proxy and retrieving a version from a proxy is zero. We denote different versions of the object by O(t). Thus, for example, if an object is generated at time t=0 and pushed directly to the client (no caches in between) who is t=T away (Fig 2), then the version of the object received by the client at time T is O(0). The client sends the request at time t=T and receives the reply instantaneously. Now as shown in Fig 3, if only the proxy at halfway between the server and the client can cache, we see that if the client sends the request at t=0, it receives the reply at t=T. The version of the object that it receives is O(0).

We see that as far as the client is concerned, there are two time points involved, namely the time at which the request is sent and the time at which the reply is received. The difference between these two time points is the latency in object retrieval and that is a parameter that the user perceives. Shorter the latency the better, as the user has to wait only a short time to see the information requested. Once the information is displayed on the screen, we should know whether it is consistent or not. That is, the consistency of the object should be based on the time of arrival of the reply and not on the time of sending the request. For example, if two users receive two different versions of the object at the same time, then the user who received the latest version has received more consistent information. Following this argument, the two cases shown in Fig 2 and Fig 3 have the same consistency as both clients receive identical versions of the object at the same time.

Now, let us consider the general case in continuous time and assume that only the proxy that is pT ($0 \le p \le 1$) away from the client can cache the object. In this case, if the client sends the request at $t=t_0$, the reply is received at $t=2pT+ t_0$

and the version of the object received is $O(2pT-T+ t_0)$. We note that the object produced by the server at $t=2pT+ t_0$ is $O(t_0+2pT)$ and therefore, the staleness of the version that the client receives is T.

In practice, the server generates different versions of an object at discrete time points (Fig 4). The time difference between two consecutive versions of the objects $O(t_i)$ and $O(t_{i-1})$ is known as the update interval (Δ_i).

As a result, if the client sends a request at $t_{sent}=t_0$, the time point of arrival of the reply is given by

$t_{rcv}=2pT+ t_0$ (1)

and the version of the object received

$O_{rcv} (t_{rcv})= O(t_c)$, (2)

where $t_c \leq 2pT-T+ t_0 < t_{c+1}$.

If we denote the version of the object residing in the origin server at $2pT+ t_0$ by $O(t_s)$, where $t_s \leq 2pT+ t_0 < t_{s+1}$, the staleness of the object received by the client is given by $t_{stale} = t_s - t_c$ (3)

In a real world scenario, there is a considerable amount of object insertion delay (the time required to cache the object) and removal delay (the time it takes to retrieve the copy) [2]. Let us denote the insertion delay by t_{in} and the removal delay by t_r. Therefore, the latency in retrieving a copy from the cache by the client has to be now modified to $2pT+ t_r$. The different versions of the object have to be shifted in time to take into account the insertion delay. The modified diagram is shown in Fig 4. In this case, the above equations (1), (2) and (3) are modified as

$t_{rcv}=2pT+ t_0 + t_r$ (4)

$O_{rcv} (t_{rcv}) = O(t_c)$, (5)

where $t_c \leq 2pT-T+ t_0 - t_{in} < t_{c+1}$

and

$t_{stale} = t_s - t_c$, (6)

where $t_s \leq 2pT+ t_0 + t_r < t_{s+1}$.

4 Analysis of Push Caching

In this section, we analyse push caching systematically using the equations (4), (5) and (6). We also define a parameter called push radius which provides the boundary for push caching.

From equations (5) and (6), we see that for a given object produced by a given source, the staleness depends on the values of p, T, t_0 , t_{in}, t_r and the update frequency of the object. If this update frequency or corresponding update time interval (t_u) is constant, from eqs (5) and (6) we can infer that, if

$t_u \geq t_r + T + t_{in}$, (7)

the staleness is either t_u or zero. In fact, whether the staleness is zero or t_u is determined by the position of $2pT+ t_0$ relative to the time series of the version of the object. On the other hand, if $t_u < t_r + T + t_{in}$, then the staleness will be more than t_u.

The inequality (7) is a very important expression as it reveals that the popular belief that the push caching always reduces staleness is not correct. By simply decreasing p, that is allowing more push caching, we cannot decrease the staleness. It is possible to achieve zero staleness even with p=1, i.e. without push caching. On the other hand, as we see from the inequality (7), if the update interval is smaller than the time delay between the server and the client, there is always staleness. If there is a number of caches between the server and the client that participates in the pushing, we have to modify inequality (7) as,

$t_u \geq T+\Sigma(t_r + t_{in})$, (8)

where the summation is taken over all the caches. This shows that the staleness worsens with the increase of the sum of object insertion delay and the removal delay, and the number of caches in between the client and the server. At the other extreme, if no server replication or caching at intermediate nodes is done (server replication is practically a form of pushing) the server load increases with the increased number of requests and ultimately server breakdown occurs. The removal delay increases with the ratio load on the processor/processor power, and because of that reason, the removal delay is high closer to the server than closer to the client. If there is server replication or intermediate caching, the load may increase at these mirror sites or at caching nodes and as a result, the object insertion delay as well as the object removal delay will increase. Therefore, if we want to reduce staleness to a very low value, we have to push the corresponding objects beyond the region of the caches that are experiencing heavy loads using a method that bypasses the cache hierarchy.

We define the push radius (R_{push}) as the distance from the origin server to the further most proxy server in the cache hierarchy along the path from the server to the client for which inequality (8) holds true. At the client, we define a similar quantity called pull radius (R_{pull}) which is the distance from the client to the further most proxy server for which the latency is less than or equal to t_{lm}, the maximum latency that the client can tolerate. If the two regions formed by push radius and pull radius touch each other or intercept, we can achieve minimum staleness for the object retrieved, by pushing the object to a cache that is common to both regions. On the other hand, if the two regions do not intercept, we have to push the object to a cache within the pull radius by increasing the push radius. As we can see from the inequality (8), this can only be achieved by bypassing all the heavily loaded caches, either using a priority scheme such as Resource Reservation Protocol (RSVP) or following an alternative route or using private lines. Pushing right through to the client's network definitely reduces the retrieval delay much, but there will be additional traffic produced, as the server does not know which clients require a certain object. A second disadvantage is that the caching server in the client's network must be of high capacity and processing power, to store and process all the objects sent by the origin servers and this should be ruled out. Therefore, for a given object, it is necessary to find an intermediate position to cache the object and the push radius and pull radius will determine this location.

In this section, we discuss the effect of randomness of the quantities involved on the staleness. We note that p, the request time point of the object, T, object insertion delay and the removal delay, and the update interval of the object are all random. If we assume that p is a constant and t_0 is fixed and all others are Poisson distributed with mean λ_1, λ_2, λ_3 and λ_4 respectively, using equations (4) through (8), we can write,

$$t_{rcv}=2p\lambda_1+ t_0 + \lambda_3 \quad (9)$$

$$O_{rcv}\,(t_{rcv}) = O(t_c), \quad (10)$$

where $t_c \leq 2p\lambda_1 - \lambda_1+ t_0 - \lambda_2 < t_{c+1}$,

$$t_{stale} = t_s - t_c \quad (11)$$

where $t_s \leq 2p\lambda_1 + t_0 + \lambda_3 < t_{s+1}$

and

$$\lambda_4 \geq \lambda_1+\Sigma(\lambda_3+\lambda_2)], \quad (12)$$

Inequality (12) tells us that, if we can approximate the sum of object insertion delay and the removal delay, the update interval of the object and the delay from the server to the client with Poisson distributions with known mean, we can minimise the mean staleness of the retrieved object. The server should take each proxy in its path to the client one by one, calculate the quantity on the right hand side of the inequality (12). If the required delay in retrieval is known, the server can find a cache within the pull radius to minimize staleness.

5 Conclusion

We have developed a new mathematical model for push caching and analysed the influence of push caching on object retrieval time and staleness. We have shown that the popular belief that staleness can be reduced by simply pushing the object from the origin server to an intermediate cache is a little illusive. In fact, if proper pushing mechanisms are not adopted or proper intermediate caches are not selected, we may increase the staleness. We have introduced a new quantity called push radius, that tells the server how far it should push an object using special methods in order to reduce the staleness close to zero. Pushing definitely reduces the retrieval time if the removal time for the copy of the object is small compared to the delay from the server to the client.

We believe that this model and the analysis will help the cache designers design web caching in a more systematic manner. We are presently investigating the possibility of server determining the push radius in a real life situation.

In the years to come, browser set-up and support will be completely automated. A typical browser will automatically find whatever resources it needs, including caches, each time it begins operation. Then, the proxy caches will be completely transparent to the browser user.

References:

[1] A Chankhunthod et al, A Hierarchical Internet Object Cache, http://catarina.usc.edu/ danzig/cache/cache.html

[2] R. Tewari, M. Dahlin, H. M. Vin and J. S. Kay, Design Considerations for Distributed Caching on the Internet, *Proceedings of ICDCS 99*, Austin, May 1999,

[3] Fan Li et al, Summary cache: a scalable wide-area Web cache sharing protocol, *IEEE/ACM Transactions on Networking*, Vol 8, Issue 3, June 2000, pp 281 –293.

[4] Hash Routing Architecture: Briefing on Super Proxy Script, http://naragw.sharp.co.jp/ sps/sps-e.html

[5] D. Wessels and K. Claffy, Internet Cache Protocol (ICPv2), http://ds.internic.net/rfc/rfc21 86 .txt

[6] P. Vixie and D. Wessels, Hyper Text Caching Protocol (HTCP/0.0), *Request for Comments RFC 2756*, http://www.faqs.org/rfcs/ rfc2756.html

[7] K.W. Ross, Distribution of Stored Information in the Web, http://www.eurecom. fr/~ross/CacheTutorial2/sld001.htm

[8] Web Cache Communication Protocol, http://www.cisco.com/warp/public/732/wccp/

[9] A. Dingle, Cache Consistency in the HTTP 1.1 Proposed Standard, http://w3cache.icm.edu. pl/workshop/talk4/

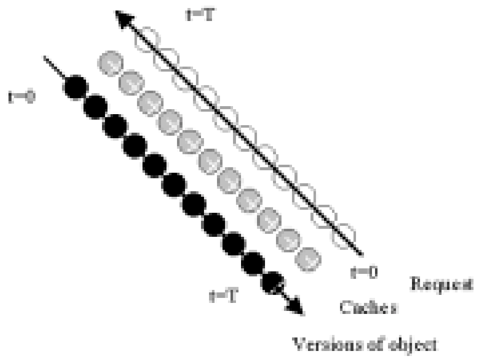

Fig 1: Continuous time representation of versions of objects and proxies

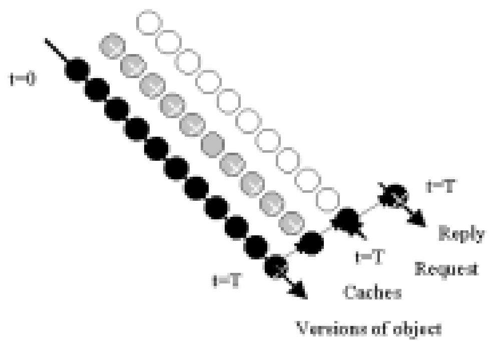

Fig2: A request fetches the object from the cache at the client

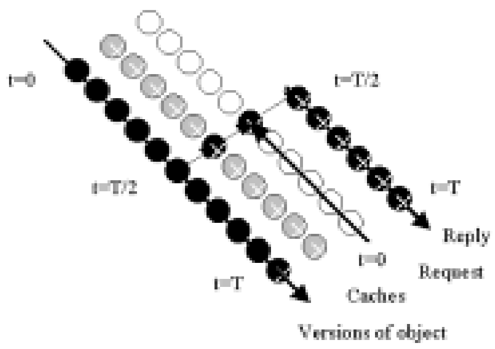

Fig 3 A request fetches the object from a cache half way between the client and the server

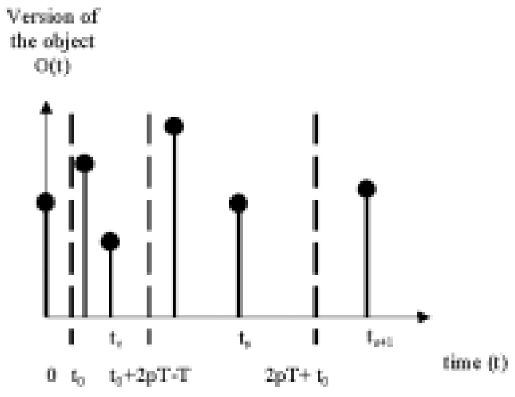

Fig 4: Time series of the versions of the object and the versions of the object in the cache and in the server

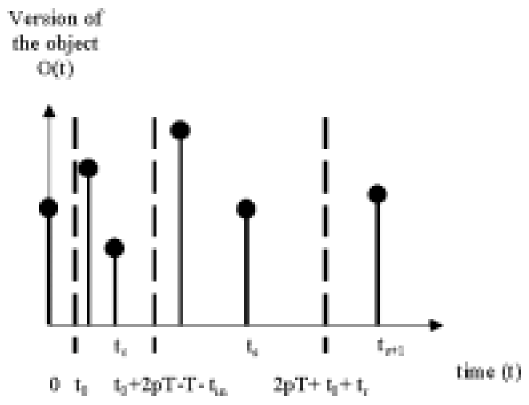

Fig 5: As in Fig 3, but object insertion delay and removal delay included

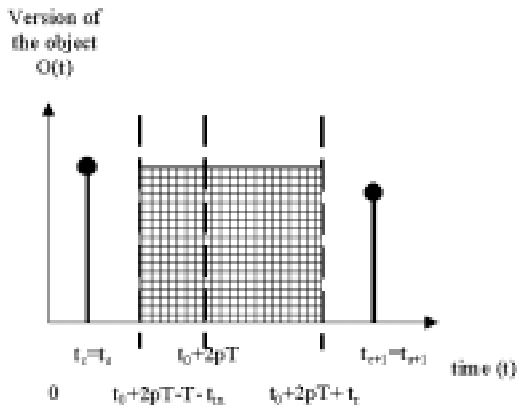

Fig 6: Region of $t_r + T + t_{in}$ if staleness is zero

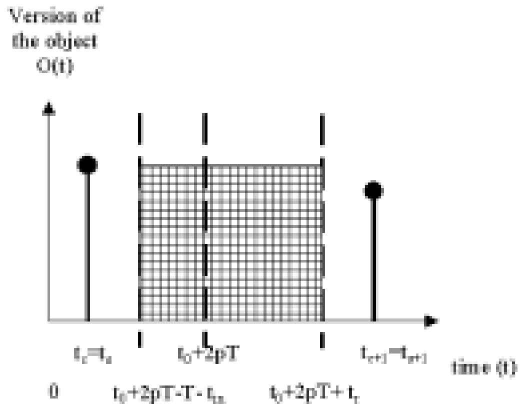

Fig 7: Region of $t_r + T + t_{in}$ if staleness is t_u