A new Approach to Decrease Invalidate Rate of Weak Consistency Methods in Web Proxy Caching

Chen Chen[1,3], Qingyun Liu[1,3], Hongzhou Sha[2,3], Zhou Zhou[1,3], and Chao Zheng[1,3]

[1] Institute of Information Engineering, Chinese Academy of Science[1]
[2] Beijing University of Posts and Telecommunications[2]
[3] National Engineering Laboratory for Information Security Technologies[3]
{chenchen,liuqingyun,zhouzhou,zhengchao}@iie.ac.cn
shahongzhou@nelmail.iie.ac.cn

Abstract. With the growing demand for accelerating large scale web access, web proxy cache is widely used. To make full use of computing resource and bandwidth of proxy cache nodes, weak cache consistency is the best choice in most cases. Traditional refreshing methods like Adaptive TTL will cause high invalidate rate of web pages. We introduce a new effective way to decrease the invalidate rate of frequently queried objects in weak consistency scheme. Based on Zipfs law, our method focuses on giving the hotspot objects more priorities during cache refreshing process, which reduces the invalidate rate on hotspot objects by paying less concentration on the less frequently queried objects.

Keywords: Web Proxy Cache, Invalidate Rate, Weak Consistency

1 Introduction

Nowadays the scale of the World Wide Web is growing explosively, and web proxy cache is widely used. Its well known that transparent proxy caches like Squid can only maintain a weak consistency. TTL (Time-To-Live), Adaptive Time-To-Live, Client Polling and Piggyback Invalidation are popular weak consistency mechanisms. TTL and Adaptive TTL mechanism use timers based on modification-time to update cached pages, making a stable invalidate rate relatively. They cannot take advantage of real-time cache accessing behavior C limiting the invalidate rates dependency on the server modification time uncontrollably. And the objects which are less of worthy in cache updating process occupy large burden on the limited bandwidth and computing resources. The client polling mechanism will update cache items periodically, but making a high invalidate rate when the updating period is too long. It will also make a heavy burden on performance when the period is set to too small. And piggyback invalidation mechanism calls for more cooperation between proxy caches and web servers, which is not applicable in transparent proxy caches. Among all of the

above weak consistency mechanisms, we found a way of improving the traditional weak consistencys performance by using the distribution of user accessing behavior on traditional effective Adaptive TTL or TTL methods. Furthermore, we also introduce an estimating method for hotness calculation, based on the exponential hotspot raising assumption.

2 Related work

Lee Breslau[1] described that the Zipfs law is implemented in the web caching area naturally. And Spanoudakis[2] made a new approach on the invalidate rate improvement, focusing on the prediction of more accurate updating timer compared with the widely adopted Adaptive TTL method. J Gwertzman[3] proposed the Adaptive TTL method, which can produce an invalidate rate less than 5% - a big improvement among weak consistent schemes. While Cao[4] made experimental comparison among Adaptive TTL, Polling-every-time and Invalidation methods, showing that invalidation method is the most recommended protocol, which could make the best performance either on network traffic and workload, or on the invalidation rate. But the invalidation method introduced more burden on the complexity between cache server and the resource server - which does not fit for the proxy cache environment, especially when the proxy cache is transparent from the content. Moreover, Li[5] developed the lease algorithm in mobile network, which can balance both space and control message usages among strong consistency strategies like polling-every-time and invalidation. Krishnamurthy[6] presented the Piggyback methods, better than many traditional methods, which gave us an pruning recommendation in algorithm selection. Barish[7] gave us an outline view of the existing web caching techniques, including proxy-caching, proxy caching and transparent caching, which are all our interests. Wang J.[8] made a remarkable description over each aspects of web caching, and introduced the weak consistency mechanism especially.

3 EW (Enhanced Weak consistency) model

The Adaptive TTL method can guarantee 5% invalidate rate at most, but with the increasing scale of the World Wide Web, 5% also means a large amount of data. As its known to all, web querying behaviors obey the Zipfs law, which shows a long tail effect on Web objects querying. In most of the implementations about web proxy caching, objects with higher frequency are more worthy of caching, which could easily cause burdens in caching computation. What we care about is to make traditional weak consistency algorithms like Adaptive-TTL more effective in the hotspot web objects and reduce the invalidate rate especially on hotspot objects, increasing the average refreshing frequency on hotspot objects in order to reduce the invalidation rate selectively. We introduce a hotness aware queue, each time when object in cache is visited, it will be operated on its measurement, and when the measurement goes over the threshold, the object will be pushed into the updating queue, which is sorted by update timer, generated

by Adaptive TTL method. Theres a watching thread outside the queue to check whether the first update time is up over and over. Whenever it is confirmed that the first object in updating queue is out of date, the object will be updated from the original Web server, the thread will also check objects form the second to the end to refresh objects. Updating methods worst time complexity is O(n). But on average therere no more computation requirements than the Adaptive TTL method. In most of the cases, Web objects' frequencies obey the Zipfs law C an exponent cumulative distribution of frequency. And in web cache environment, we should consider more characteristics of web objects such as web access behaviors including recency, frequency and web objects properties including size, cost or latency and type of objects etc. Based on the unbalanced distribution, therell be less than 40% of the objects attending the updating process experimentally, and with a tolerance on the invalid rate on the objects that are less worthy of caching, the global invalidate rate of Adaptive-TTL will reduce to 2% at most. First of all, we are going to talk about the qualification measurement of judging whether an object is worthy of caching or not.

3.1 Summarize of Frequency measurement

One of the core jobs of hotness estimation is the appropriate measurement of objects characteristics. We should give objects with larger frequency bigger priorities. Our implementation is to use the frequency information in the exponential assumption later talked about.

3.2 Types of Web Objects

From qualitative analyses of the modification periods of web objects, we conclude an experiential frequency list below:

Table 1. Experiential Frequency level of different object types

Type \ Size	Small	Large
HTML	Most	Moderate
Script	Moderate	Moderate
Other Text	Moderate	Least
Binary	Least	Least

The most frequently modified objects are more likely to be those web objects with less modification cost, such as the text formatted object like HTML. And the binary objects like flash, voice and video objects are the less likely to be changed - the modifications on such objects are usually redirected to new addresses. Moreover, the type of the objects can easily be recognized by the URL suffixes or header information of the objects. We could give each type a weight separately related to the modification frequency above.

3.3 Cost and Size of Web Objects

As for the modification cost of large objects is bigger than that of the smaller ones, we estimate that the frequency pattern and the size of objects have inverse relationship. Since it is in the proxy cache environment, internet transmission latency is another important pattern, which not only has relationship with the size of object, but also related to the delay of each objects. We describe the delay by the variable cost, and the frequency of modifications will be directly proportional to $\frac{cost}{size}$. So, we concluded all of the characteristics above to a compound measurement variable a, which is showed below:

$$a = \left(\frac{cost}{size}\right) \cdot weight \tag{1}$$

3.4 Hotness estimation model

To simplify the hotness model, we make an assumption that all of the querying hotness are similar to an exponential function exponentially as:

$$C\left(t\right) = a^t \tag{2}$$

Where "a" is the compound measurement talked above and "t" is the time. The real-time frequency could be described as $\frac{count}{\Delta t}$, where Δt is the querying period and the count is sampled during the period Δt. Since our goal is to estimate the expected hotness, which could be estimated with Δt, we should give an expression of the variable Δt. Based on assumption, we believe that each time when we make a sampling during the period, the accumulated count has an exponent relationship with t_0 as (3):

$$\Delta t = \log_a \left(1 + \frac{\Delta C_{expect}}{C\left(t_0\right)}\right) \tag{3}$$

In equation (3), variable a shows the characteristics of the web objects, t_0 is the query time since the appearance of the cached item, and $C(t)$ is the expected accumulated hit count. Δt is the expecting querying period, and C_{expect} is the expected count sampled in last period while $C(t_0)$ is the last accumulated count. Since we should give an expression of the hotness with real-time sampled frequencies, we will simplify the relationship between t_0 and Δt into Δt-only updating function with the equation below:

$$t_0 = log_a \frac{\Delta C}{a^{\Delta t} - 1} \tag{4}$$

Plugging (4) into (3), the next expected querying cycle $\Delta t'$ is generated in (5) .

$$\Delta t' = log_a \left(1 + \frac{\Delta C_{expect}}{\Delta C} \cdot \left(a^{\Delta t} - 1\right)\right) \tag{5}$$

Our curiosity is placed on the period, where we can gain the hotness information from it. The hotness could be described either as $\Delta t'$ with a descending

comparing order or as $\frac{1}{\Delta t'}$ with an ascending order. Moreover, the promotion threshold assignment could be based on the formula, too.

3.5 Hotness Aware Queue algorithm

Hotness is measured by the hotness pattern, which could be simplified as frequency. Whenever an object is queried, it will be updated in cache and be checked whether it has the qualification to promote to the Hotness Queue or not. The Hotness maintenance logic is described below:

Algorithm 1 Hotness Maintaining

Input: The key of the target object K_t
Output: The content of the object D_t
if K_t is in the Cache **then**
  Push K_t into Cache with initial Frequency
  return Not found
else
  if Object K_t exceeds Threshold **then**
    Push K_t into Updating Queue
  end if
  Update K_t's Frequency
  Find D_t in Cache with K_t
end if
return D_t

When an object is put into the updating queue, there should be an updating procedure, described as:

Algorithm 2 Updating

Initialize the Updating Queue
while True **do**
  if the 1^{st} Object out of date **then**
    Update the 1^{st} object
    for $i = 2$ to $Size(Queue)$ **do**
      if the i^{th} Object out of date **then**
        Update the i^{th} object
      else
        break
      end if
    end for
  else
    Sleep(timer)
  end if
end while

In the process, the working thread will check the Hotness Queue over and over until the main process is terminated. The updating period in hotness queue is given by outside Adaptive TTL method or traditional TTL method optionally, which are not our emphasis. In the next section, we will discuss the invalidate rate of this method, and later we will show the efficiency of data reduction based on the distribution.

3.6 Analysis

For most websites, the modification cycle of web object is approximate to constant. Let the modification cycle be T and the refreshing cycle of cached object be Δt, the invalidate rate of each cached objects could be estimated by (6).

$$rate_{invalid} \leq \begin{cases} \frac{\Delta t}{T}, \Delta t < T \\ \frac{T}{\Delta t}, \Delta t \geq T \end{cases} \tag{6}$$

According to Zipfs law, we use a distribution expression formula shown in (7), where "i" is the ranking of cached item by frequency, N is the number of all cached objects, and F_i is the relative frequency and hotness of object "i". Since the formula is uniformed, F_i can also be treated as appearance probability of objects.

$$F_i = \frac{1/i}{\sum_1^N 1/n} \approx \frac{1}{0.96i \cdot \ln 2.22N} \tag{7}$$

Let the Threshold of objects with more worthy be T, the expected number of refreshed objects N_c is described as:

$$N_c \leq \frac{1}{0.96T \cdot ln2.22N} \tag{8}$$

Because of the normalization operation in (7), Nc is also the data reduction scale, and $(1 - N_c)$ of the cached data will not be updated on average, which makes an improvement on performance. Hotter cached items will get smaller refreshing cycle than the original ones, which in fact makes the cache updating process more effective for the hotspots based on the adjusting to the real-time querying behavior. With the same refreshing computing resources, hot objects will get more refreshing opportunities, making a reduction on invalidate rate as well. Meanwhile, the optimization also reduces the wastes of computing power on the objects which are less worthy of caching.

4 Experiment

The dataset is a 30 days web querying log of 56,374,164 lines from a web proxy server, with 16,625,621 lines could be cached, of which 2,705,302 lines are unique. We made a simulation based on the data. With the simulation environment, we

can continuously test our algorithms. Our experiment are focused on the data reduction scale of cached data, which has an direct proportion to the reduction rate of invalidate rate.
Since the distribution of data in small cache is distinguished from the whole dataset distribution, especially for the scale of low frequency objects, we make the experiment between long-term querying in total 30-days and the short period distributions in 5 minutes. The distribution is showed below:

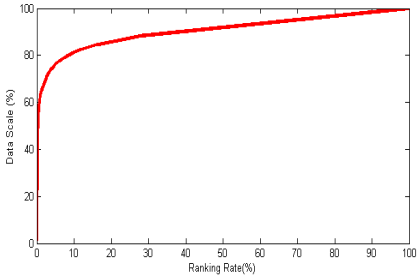

Fig. 1. 30-Day CDF

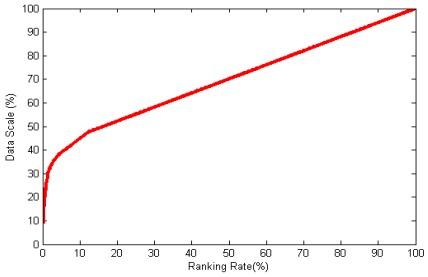

Fig. 2. 5-Min CDF

Table 2. Top-4 Least Frequently Accessing CDF Comparison

Frequency	30 Day Count	Percentage	5 Min Count	Percentage
1	1,962,470	11.80%	628	59.81%
2	304,530	3.66%	77	14.67%
3	114,184	2.06%	29	8.29%
4	63,925	0.38%	21	8.00%

Obviously, we can figure from the experiment that the reduction on data scale by frequency is more effective in short periods, with a large amount of data with frequency 1. And objects who are put into cache with less frequency are not likely to be accessed again. In the experiment , there was a 60% reduction by limiting the frequency to 1, where with a higher threshold and more invalid tolerance on less frequent items, we could either gain lower computing power on updating process or the improvement on the performance of invalidate rate C reducing 2.5 or more times of invalidate rate, or achieving a reduction of computing power on refreshing procedures.

5 Conclusion

Weak consistency caching method is one of the most important parts of proxy caching, especially for the web proxy caching area in the big-data world. Al-

though Adaptive-TTL method shows a stable performance on invalidation rate control, when it comes to the unlimited high speed network flow field, 5% is not a small rate anymore, which is hard to tolerant. Facing the problem of optimizing the invalidate rate of cache weak consistency scheme, we present an EW(Enhanced Weak Consistency) model to optimize the invalidate rate of Adaptive TTL, TTL method and so on based on Zipfs law, decreasing the cache refreshing cycle selectively. Our work mainly focuses on the measurement of web objects, hotness estimation modeling and the invalidate rate estimation. Finally, under the same refreshing calculation resources, we made a reduction of the updating data scale, reducing the invalidate rate of hotspots to about 2% when we set the promotion threshold to 1. With our work, we can use limited cache spaces and computing power more rationally and controllable. In the next process, we will optimize the qualification function and modify the model to fit for cache replacement algorithms and cache prefetching algorithms, using schedule techniques to make the updating measurements more compatible.

6 Acknowledgement

This work was supported by the National High-Tech Research and Development Plan 863 of China under Grant No. 2011AA010703, the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDA06030200, and the National Natural Science Foundation under Grant No. 61070026.

References

[1] Breslau, L., Pei Cao, Li Fan, Phillips, G., Shenker S.: Web caching and Zipf-like distributions: evidence and implications. INFOCOM '99, Vol. 1, 126-134 (1999).

[2] Spanoudakis, Manos, Dimitris Lorentzos, Christos Anagnostopoulos, Stathes Hadjiefthymiades: On the use of Optimal Stopping Theory for Cache Consistency Checks. In IEEE Informatics (PCI), 2012 16th Panhellenic Conference, 327-332 (2012)

[3] J Gwertzman, James, Margo I. Seltzer: World Wide Web Cache Consistency. In USENIX Annual Technical Conference, 141-152. (1996).

[4] Cao, Pei, Chengjie Liu: Maintaining strong cache consistency in the World Wide Web. Computers, IEEE Transactions on 47, no. 4, 445-457 (1998).

[5] Li, Xiaoqian, Feng Qiu, Huachun Zhou, Hongke Zhang, and Ilsun You: Maintaining strong consistency for the identifier-to-locator mapping cache. In Globecom Workshops (GC Wkshps), 2012 IEEE, 986-991 (2012).

[6] Krishnamurthy, Balachander, and Craig E. Wills: Study of Piggyback Cache Validation for Proxy Caches in the World Wide Web. In USENIX Symposium on Internet Technologies and Systems, 38 (1997).

[7] Barish G, Obraczke K. : World wide web caching: Trends and techniques. Communications Magazine, IEEE, 38(5), 178-184 (2000).

[8] Wang J.: A survey of web caching schemes for the internet. ACM SIGCOMM Computer Communication Review, 29(5), 36-46 (1999).