3M2-IOS-3b-8

Improved Web Cache Replacement Policy Using Web Usage Data

Sorn Jarukasemratana
Tokyo Institute of Technology
W8-59 2-12-1 Ookayama, Meguro
Tokyo, 152-8552 Japan
sorn.jaru@ai.cs.titech.ac.jp

Tsuyoshi Murata
Tokyo Institute of Technology
W8-59 2-12-1 Ookayama, Meguro
Tokyo, 152-8552 Japan
murata@cs.titech.ac.jp

Web caching is one of the fundamental techniques for reducing bandwidth usage and download time while browsing the World Wide Web. In this research, we provide an improvement in web caching by combining the result of web usage mining with traditional web caching technique. Web cache replacement policy is used to select which object to be removed from cache when the cache is full and new object should be put into the cache. There are several techniques which select the object to be removed, such as the size of the object, the number of times the object was used, or the time that the object is added into the cache. However, the flaw in those approaches is that each object is treated separately without considering the relation of those objects. We have developed a system that can record users' browsing behavior at resources level. By using information gathered from this system, we can improve web cache replacement policy so that number of replacement in cache is reduced.

1. Introduction

In recent years, the internet has become the most important tool for communication and interaction among people resulting in the increasing of data bandwidth. Web caching is well known strategy for improving the performance of web based system. Web caching can improve the performance of the WWW by creating the duplication of popular objects and temporary storing them in cache storage near the users. If those objects are requested again by the users, users will receive those objects from the cache instead of the original servers. This is called "hit" or "cache hit". Web caching gives benefits to both web users and content owners because 1) caching reduces total bandwidth usage, 2) caching reduces web page load time and 3) caching reduces loads on web site server [1]. Web caching can be applied at original server, proxy server or client-side machine. In this research, we focus on client-side caching (or browser caching) because client-side caching is more economical and effective than server or proxy caching [2].

Since cache storage has limited space, as users continuously browsing the internet, cache storage will eventually become full. When a new object need to be stored in the cache while the cache is full, cache replacement policy will determine which object will be removed to make enough space for the new object. To use the limited cache space in the most efficient way, objects that will not be used again should be removed from the cache first.

There are many cache replacement policies, each with their own algorithms for selecting object to be removed. The general goal of cache replacement policies is to increase cache hit rate. The most well-known algorithms are LRU (least recently used) and LFU (least frequency used). LRU chooses object to be removed based on the last time the object is used while LFU chooses object based on how many times the object is used. Others properties are used in others algorithms such as size of the object, total time used for download the object, or the last modification time of the object. There are also many algorithms that use more than one attributes to choose the object to be removed such as Hyper-G [3] which uses the combination of frequency, recency and size of the object.

Each algorithm has its own advantages over others. For example, in the case that system has sufficient processing power and memory resources, complex algorithms which require more computation are more suitable. On the other hand, on a system with limited processing and memory resources, randomized strategy is preferred because it requires less memory and computation [4].

Web usage mining is one of the interesting topics by many researchers since the WWW becomes a part of our daily life. Web usage mining is the process of extracting useful information from users' browsing history. The benefit of web usage mining is to let researchers understand the behavior of internet users and use that knowledge to improve their web browsing experiences. This technology enables web sites to be personalized for each individual user. One of the successful examples of web usage mining is real-time recommendation system, such as Youtube.com recommended VDO or Amazon.com book recommendation.

In this research, we incorporate web usage mining data into the cache replacement policy to create a new algorithm that can perform better than existing algorithms. Proposed algorithm is a hybrid algorithm based on recency, frequency and users' web usage history. The idea of this algorithm comes from the fact that nowadays users tend to visit the same set of websites every day. Objects from those websites should be prioritized and should stay in the cache longer than objects from other websites.

2. Related work

2.1 Web caching strategies

Web caching had become a hot topic among researchers since Luotonen and Altis [5] introduced proxy server to research field in 1994. There are at least 3 survey papers on web cache replacement algorithms. A survey of web cache replacement

Contact: Sorn Jarukasemratana, Tokyo Institute of Technology, W8-59 2-12-1 Ookayama, Meguro Tokyo, 152-8552 Japan, sorn.jaru@ai.cs.titech.ac.jp strategies is presented by Podlipngi and Böszörmenyi [1] in 2003. They classified replacement strategies into 5 classes: recency-based strategies, frequency-based strategies, recency/frequency-based strategies, function-based strategies, and randomized strategies. Recency-based strategies use temporal factor to manage the cache. Basically, the least recently referenced object will be removed from the cache. Recency-based strategies are adaptive to popularity change and mostly require low overhead. However, these strategies usually place too much emphasis on recency factor alone, which is the disadvantage of recency-based strategies. Frequency-based strategies use frequency as their main factor. Frequently called objects or popular objects tend to be kept in the cache longer than unpopular objects. These strategies perform very well at caching objects in the environment that popular objects do not change frequently. However, on quickly changing environment, frequency-based strategies will perform poorly. Recency/frequency-based strategies use both recency and frequency properties to find objects to be removed. Many strategies in this class also use other factors in their algorithms. The combination usually bring better results but at the cost of overhead and complexity. Function-based strategies use general function to calculate scores for all objects in the cache then object that has the least score will be removed. Many parameters such as size, recency, or frequency are used in the function. The strongest advantage of these strategies is that no particular attribute is dominant. However, these strategies create the largest overhead and complexity among all classes. The last class is randomized strategies. The goal of these strategies is to reduce the complexity and overhead. In result, these strategies are very simple to implement. The problem of these strategies is the evaluation. Different simulations on same test data set can give different results.

Wong [4] stated that there are more than 50 cache policies by the year of 2006 which is the year he published his cache replacement policies review. Instead of arguing about which cache replacement algorithms are better, he stated that each algorithm will perform better than others in their favored environment. For example, frequency-based strategies perform well when popular objects are not changing rapidly. Romano and ElAarag [6] did a quantitative study of web cache replacement strategies by comparing 19 different strategies running with the same data sets.

As stated by Wong in his research, cache replacement policies are heavily researched in the past. Newer researches in this topic are mostly conduct by using knowledge from another research fields. Tirdad et al. [7] used genetic algorithm and genetic computation to create a model for cache replacement policy. Ali and Shamsuddin [8] used neuro-fuzzy system to create an intelligent web caching scheme. Torkzaban and Rahmani [9] used multi expert technique to create a cache system that can select the best cache policy depending on the environment. Geetha et al. [10] created SEMALRU, a LRU based algorithm that uses semantic data of the web pages.

2.2 Web usage mining

Web usage mining is the process of extracting useful information from users' browsing history. Srivastava et al. [11] offered a taxonomy for web usage mining in 2000 since then web usage mining has been given more interest in web mining field. Web usage mining can be done at 3 levels: server, proxy server, or client-side machine. In an early stage, web usage mining usually done by analyzing server logs getting from severs or proxy servers, such as the work of Myra Spiliopoulou [12] where web usage mining data is used for evaluating web sites. Client-side log mining is getting more popularity because of the technological advances such as web browser plugins that allows researchers to collect data directly from users.

Studies of client-side logs can give many insights about users' behaviors. Zhou et al. [13] proposed a temporal-based web access behavior which focuses on time of the day that users access the web. Khoury et al. [14] created a graph based on users' activities for some period of time. The result is that they can identify users' behaviors such as using Wikipedia or e-mail as hubs for traversing to other websites or finding frequent traverse paths of users.

3. Algorithm

Our algorithm is based on LRU (Least Recently Used) strategy but with extra properties. Algorithm is divided into 3 parts: cache storage structure, cache insertion policy, and cache eviction policy. The main idea behind this algorithm is based on the result of Khoury et al. and the result of our own usage mining data that users tend to visit a same set of websites every day. Khoury et al. reported that Wikipedia sites and e-mail sites are the hub of users browsing sessions, while in our own data, social media websites like Facebook and news websites are the main hub. Even our algorithm is based on LRU, our algorithm falls into Recency/frequency-based strategies class on Podlipngi and Böszörmenyi classification because we also incorporate frequency-based properties into our algorithm.

3.1 Cache storage structure

Like other LRU based strategies, our cache objects (such as pictures, Java script files, html files) are kept in a list. Each object in a list contains many properties. There are 6 properties, object id, next object id, previous object id, size of the object, object's host, and object's HP (algorithm generated integer value). Object's host is the URL of the server that provides users with this object. This is the extra information gained from web usage mining. In normal browsing situation, this information is omitted from the users. However, with network tools, it is possible to record this information while users are browsing the internet. Object's HP stands for object's hit-point. This integer value is generated and managed by the algorithm. Objects with less HP are considered unimportant and will be removed first if eviction occurs. The maximum HP value need to be defined to prevent objects from being too popular and will never be evicted from the cache. The minimum HP value is 0.

Apart from objects storage, this algorithm needs to keep at least 2 lists of unique host URL: today's host list and yesterday's host list. The size of the list is very small compare to objects list because many objects share the same host URL. These lists take part in determining the HP of the objects. The lists are maintained on daily basis.

Compared with LRU algorithm, our algorithm requires more space since ours has to store lists of unique hosts and for each object in the objects list, our algorithm has more properties.

3.2 Cache insertion policy

When new object wants to be entered the cache, the following algorithm (Figure 1.) is applied.

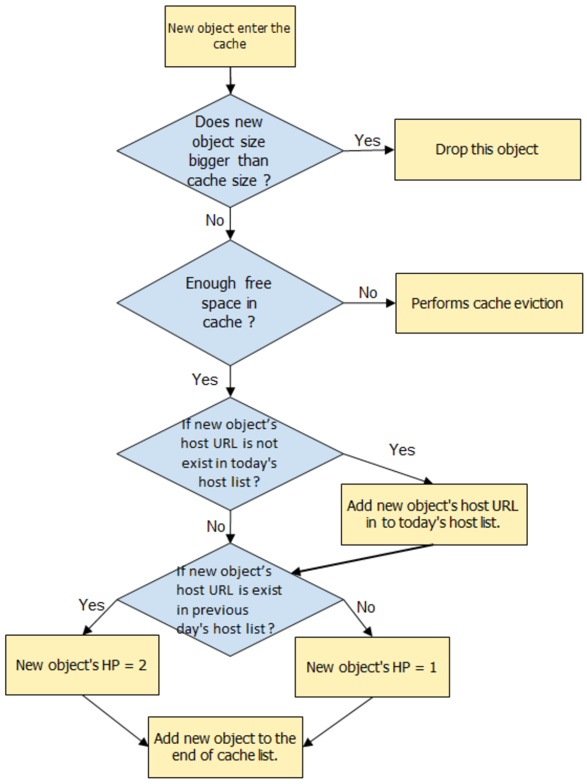

New object enter the cache
Does new object size bigger than cache size ?
Yes
Drop this object
No
Enough free space in cache ?
No
Performs cache eviction
Yes
If new object's host URL is not exist in today's host list ?
Yes
Add new object's host URL in to today's host list.
No
If new object's host URL is exist in previous day's host list ?
Yes
No
New object's HP = 2
New object's HP = 1
Add new object to the end of cache list.

Figure 1. Cache insertion flowchart

The last condition in flowchart is the important point of this algorithm. This means objects are given more hit-point if they are related to previous browsing history. The second condition from the last is for collecting users' history for use later. If object entering the cache is not new object (cache hit occurs), the hit object will be placed at the end of the list and its HP will increase by 1.At 24 hours interval, today's host list will be copied to previous day's host list. Then today's host list will be emptied. It is also possible to have more than one list of previous host URL.

LRU insertion policy is quite different from ours since we have more properties (HP and host URL) and a host list to maintain. Also normal LRU based algorithm will insert new object at the front of the list while remove the last object in the list (oldest object), our algorithm puts new object at the back of the list instead. The reason is because our algorithm eviction process is quite different in the eviction process of LRU.

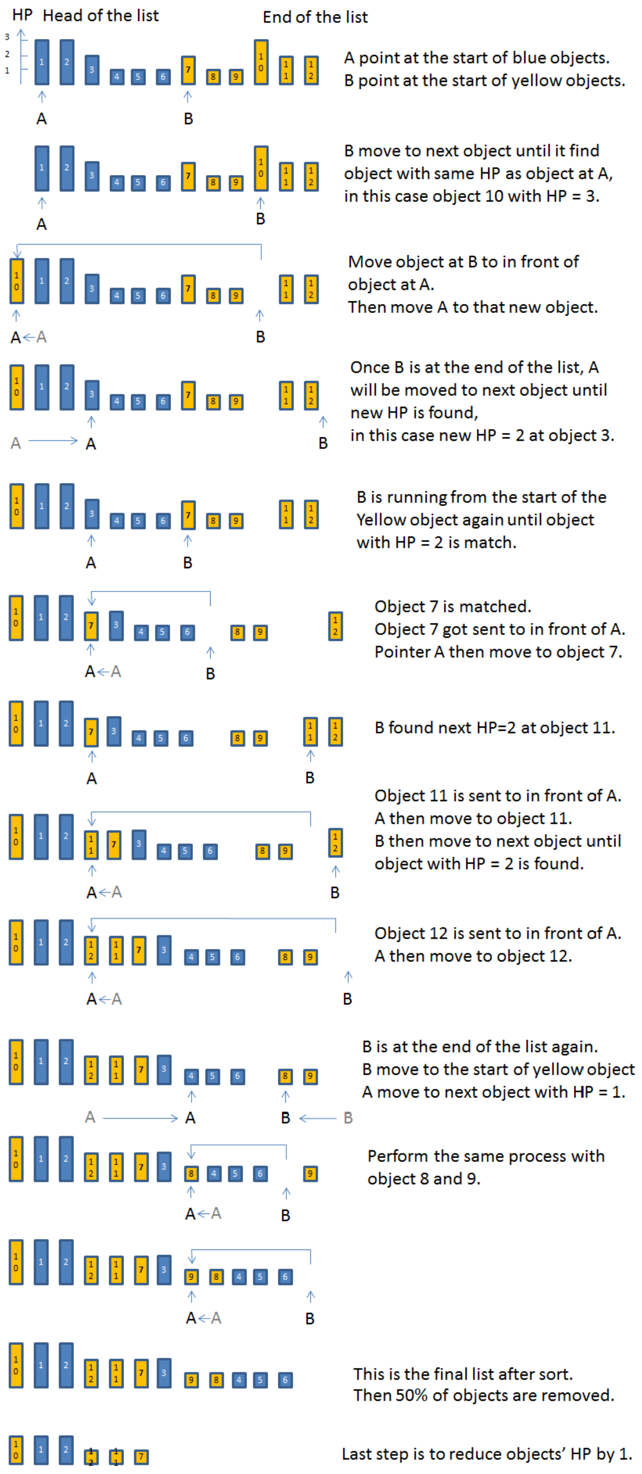
3.3 Cache eviction policy

Once the cache is full and new object need to be put in, eviction process will be called. The process works as followed.

1. Sort object list from high HP to low HP, If HP is the same newer object will be placed in front of older object.
2. Remove objects from the cache, starting from the end of the list (the less HP end) until X% of cache space is available.
3. Reduce HP of all objects in the list by 1.

The eviction process is explained with an example shown in Figure 2. Assume that

- There are 12 objects in our cache, represent with rectangle.
- Object 1 is at the head of the list while object 12 is at the end of the list.
- Object height refers to HP of that object. Objects 1, 2, and 10 have 3 HP. Objects 3, 7, 11, and 12 have 2 HP. Objects 4, 5, 6, 8, and 9 have 1 HP.
- Blue objects are sorted from previous eviction process.
- Yellow objects are new objects or objects that got "cache hit" and got moved to the end of the list.



A point at the start of blue objects.
B point at the start of yellow objects.

B move to next object until it find object with same HP as object at A, in this case object 10 with HP = 3.

Move object at B to in front of object at A.
Then move A to that new object.

Once B is at the end of the list, A will be moved to next object until new HP is found,
in this case new HP = 2 at object 3.

B is running from the start of the Yellow object again until object with HP = 2 is match.

Object 7 is matched.
Object 7 got sent to in front of A.
Pointer A then move to object 7.

B found next HP=2 at object 11.

Object 11 is sent to in front of A.
A then move to object 11.
B then move to next object until object with HP = 2 is found.

Object 12 is sent to in front of A.
A then move to object 12.

B is at the end of the list again.
B move to the start of yellow object
A move to next object with HP = 1.

Perform the same process with object 8 and 9.

This is the final list after sort.
Then 50% of objects are removed.

Last step is to reduce objects' HP by 1.

Figure 2. Cache eviction algorithm example

In LRU, once a cache is full, insertion and eviction occurs almost always together because LRU only evicts as less objects as possible to maintain the cache to be near full all the time. This can increase the hit chance because there are many objects in the cache. However, in our algorithm we already determined that objects that have less chance of a hit are at the end of the list. Therefore, it is safe to remove many of them at once. Moreover, with this HP system, new objects need to stay in the cache for sometimes in order to gain more HP value. If eviction process occurs often, all new objects are likely to be removed.

The final step of reducing all objects HP by 1 is crucial in this algorithm. In case of a popular object with maximum HP (assume that maximum HP is 5) that suddenly not get hit again. This object will be reduced to 0 HP in 5 eviction processes. Object that has 0 HP is guaranteed to be removed from the cache because new object is starting with 1 HP.

Our algorithm has some configurable parameters which can be adjusted.

- Maximum HP – high maximum HP results in a longer stay of the object cache (if it can manage to reach that HP). In a data set that popularity of objects is changing regularly, maximum HP should be set to a low value.
- X% of eviction – higher X value means longer time before cache need to perform eviction process again. However, higher X has higher risk of losing popular objects.
- Number of previous host list – in this experiment we used only 1 list which is yesterday's list. However, it is also possible to keep record of several lists such as weekly list or monthly list and award objects that share host URL accordingly.

4. Experiment

To evaluate our algorithm, first we collected raw web usage data from 4 volunteers from different occupations which are entrepreneur engineer programmer and student. All of the volunteers use internet regularly both for work and personal life. Data collection lasted from 3-9 days, depend on the usage of each volunteer. There are 2 main reasons we decided to collect data by our own, instead of using existing trace data (raw web usage data) that available in the internet. The first reason is that our algorithm requires host data. Most trace data that are available for download does not provide this property. The second reason is that most trace data are very old and possibly out dated. For example, the latest client-sided trace data from The Internet Traffic Archive [14], is collected in 1996 or the latest client-sided trace data from web-caching.com [15] is dated back in 1998. The WWW is changing all the time especially the trend of social networking website such as Facebook. In our trace data, Facebook appears very often as both page hit and as web hub (users use this website as a start point to link to other website).

Data collection was performed by installing Mozilla Firefox and Mozilla Firefox plugin on client machines. Special configurations of Mozilla Firefox are needed to be adjusted. Most cache functions of web browser were disabled in order for plugin to collect accurate browsing data. Volunteers can use Firefox as normal web browser while add-on collects browsing data in the background. There are many drawbacks in this method. 1) Browsing data are highly confidential data. We have transformed all raw data into numerical number with irreversible method to keep privacy level as high as possible. 2) Volunteers were forced to use Mozilla Firefox. The reason we choose this browser in our research is from Firefox rich add-on functionality. 3) Since all cache functions were disabled during the data collecting period, browsing speed was reduced due to no cache. 4) Unlike other data gathering methods such as performing a task or answering questionnaires, data collection lasts for several days. The task is a burden for volunteers.

HTTP data was collected in XML format. One HTTP request contains many information about data and transaction. One request is for one object which means if web page contains 20 pictures and 1 html file, there will be 21 requests for that page. Example of HTTP request can be seen in Figure 3.

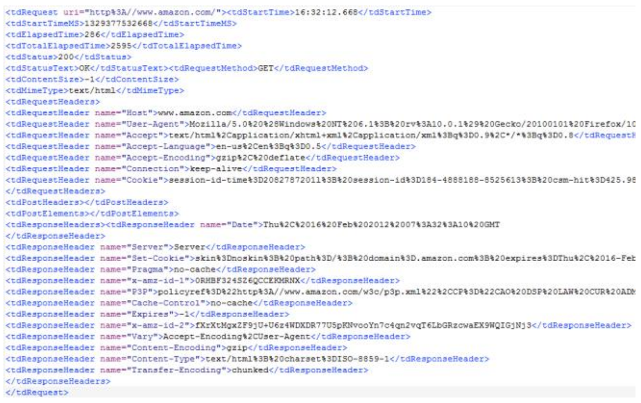

```
<tdRequest uri="http%3A//www.amazon.com/"><tdStartTime>16:32:12.668</tdStartTime>
<tdStartTimeMS>1329377532668</tdStartTimeMS>
<tdElapsedTime>286</tdElapsedTime>
<tdTotalElapsedTime>2595</tdTotalElapsedTime>
<tdStatus>200</tdStatus>
<tdStatusText>OK</tdStatusText><tdRequestMethod>GET</tdRequestMethod>
<tdContentSize>-1</tdContentSize>
<tdMimeType>text/html</tdMimeType>
<tdRequestHeaders>
<tdRequestHeader name="Host">www.amazon.com</tdRequestHeader>
<tdRequestHeader name="User-Agent">Mozilla/5.0%20%28Windows%20NT%206.1%3B%20rv%3A10.0.1%29%20Gecko/20100101%20Firefox/1C
<tdRequestHeader name="Accept">text/html%2Capplication/xhtml+xml%2Capplication/xml%3Bq%3D0.9%2C*/*%3Bq%3D0.8</tdRequestH
<tdRequestHeader name="Accept-Language">en-us%2Cen%3Bq%3D0.5</tdRequestHeader>
<tdRequestHeader name="Accept-Encoding">gzip%2C%20deflate</tdRequestHeader>
<tdRequestHeader name="Connection">keep-alive</tdRequestHeader>
<tdRequestHeader name="Cookie">session-id-time%3D2082787201l%3B%20session-id%3D184-4888188-8525613%3B%20csm-hit%3D425.98
</tdRequestHeaders>
<tdPostHeaders></tdPostHeaders>
<tdPostElements></tdPostElements>
<tdResponseHeaders><tdResponseHeader name="Date">Thu%2C%2016%20Feb%202012%2007%3A32%3A10%20GMT
</tdResponseHeader>
<tdResponseHeader name="Server">Server</tdResponseHeader>
<tdResponseHeader name="Set-Cookie">skin%3Dnoskin%3B%20path%3D/%3B%20domain%3D.amazon.com%3B%20expires%3DThu%2C%2016-Feb
<tdResponseHeader name="Pragma">no-cache</tdResponseHeader>
<tdResponseHeader name="x-amz-id-1">0RMBF324SZ6QCCEKMRNX</tdResponseHeader>
<tdResponseHeader name="P3P">policyref%3D%22http%3A//www.amazon.com/w3c/p3p.xml%22%2CCP%3D%22CAO%20DSP%20LAW%20CUR%20ADM
<tdResponseHeader name="Cache-Control">no-cache</tdResponseHeader>
<tdResponseHeader name="Expires">-1</tdResponseHeader>
<tdResponseHeader name="x-amz-id-2">fXrXtMgxZF9jU+U6z4WDXDR77U5pKNvooYn7c4qn2vqT6LbGRzcwaEX9WQIGjNj3</tdResponseHeader>
<tdResponseHeader name="Vary">Accept-Encoding%2CUser-Agent</tdResponseHeader>
<tdResponseHeader name="Content-Encoding">gzip</tdResponseHeader>
<tdResponseHeader name="Content-Type">text/html%3B%20charset%3DISO-8859-1</tdResponseHeader>
<tdResponseHeader name="Transfer-Encoding">chunked</tdResponseHeader>
</tdResponseHeaders>
</tdRequest>
```

Figure 3. HTTP Request

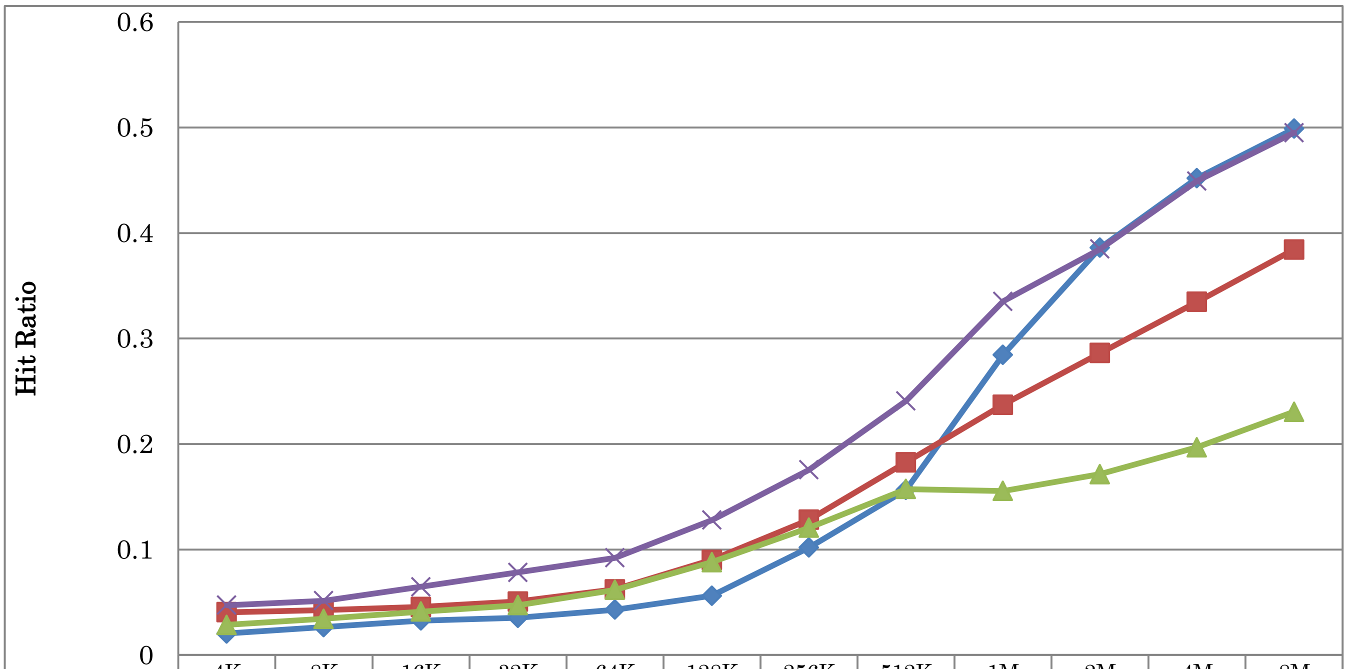
There were total of 780 megabytes of raw log data with more than 234,425 HTTP requests. We compressed the data into numerical number so that it is not irreversible and easier to compute in the simulation. The example of compressed data for one transaction is "1 3510 47551 0 0 0 221 3502 375 316894181".

We tested our algorithm and other 3 algorithms which are LRU, SIZE, and HYBRID. LRU is the baseline of recency-based algorithm. SIZE algorithm emphasizes on size of the object, while HYBRID is a mixed strategy between recency, frequency, and size. The parameters of our algorithm were X=50, maximum HP is 5 and we use 1 host list which is yesterday's list. We run all algorithm on different cache storage size, range from 4KB to 8MB. The result is shown in Figure 4. X axis refers to hit-rate. Y axis refers to cache storage size. Hit-rate translates directly to the performance of the cache algorithms, which means the higher the better.

5. Evaluation and Discussion

Our cache replacement algorithm performed better than baseline algorithm LRU at various cache sizes which are from 4KB up to 1MB. From 1MB both algorithms performances start to converge and at 2MB performances of both algorithms are converted into same value. Our algorithm also outperformed Size and Hybrid at all cache storage size.

The reason our algorithm performs better than LRU at lower cache size is because popular objects did not get called fast



	4K	8K	16K	32K	64K	128K	256K	512K	1M	2M	4M	8M
LRU	0.020439	0.026451	0.032683	0.035288	0.042927	0.056235	0.10181	0.156571	0.284496	0.386062	0.451966	0.49874
Size	0.040627	0.042554	0.045569	0.050719	0.062174	0.090098	0.12817	0.182625	0.237294	0.286209	0.334805	0.384319
Hybrid	0.028524	0.034402	0.041233	0.047239	0.061752	0.087964	0.12058	0.15733	0.155428	0.171366	0.196936	0.230567
New	0.047178	0.051532	0.064681	0.078362	0.09216	0.127827	0.175494	0.240927	0.335258	0.38482	0.449184	0.494973

Figure 4. Hit Ratio vs. Cache Size of various cache algorithm

enough before it got evicted from LRU cache. On the other hand, our strategy can give more priority to popular objects by learning from the past (via previous day host list). However, when the cache size is big enough, the performances of both algorithms are the same. The reason that guarantees our algorithm not to be worse than LRU in performance is because we incorporate LRU into our strategy.

Comparison of our algorithm against LRU is as follows.

Advantages

- Our algorithm performs better at low storage space
- Once cache is full LRU will perform cache eviction process all the time while our strategy performs once in a while. If counting overhead for files I/O, even total files transfer is the same, our algorithm will create less overhead.

Disadvantages

- Our algorithm is more complex in both structure and calculation. Space overhead of our algorithm is for storing host property and HP property for every object in cache list, plus space for storing unique host list. CPU overhead is for calculation of HP value and sorting list when eviction occurs.

Our algorithm can be fit perfectly for web browsers on smartphones. Since EDGE and 3G, browsing internet via mobile devices is very common. Our algorithm is very suitable for low-end smartphones where memory is very limited. For example, one low-end Android based smartphone Sumsung Galaxy Ace contains 158MB of RAM. RAM on smartphones is shared by its operating system, web browser, and other always-on application such as 3G or GPS making RAM very precious resource. In this case, by giving away only 1MB of RAM for web caching, 24% of objects will receive cache hit. Moreover, smartphone market now is very competitive about CPU speed while memory is mostly neglected, which can compensate our CPU overhead well.

6. Conclusion

In this research, we have proposed a web cache replacement policy based on recency-based algorithm and users' web usage data. Our algorithm can perform significantly better than baseline LRU algorithm in low cache storage environment and perform the same as LRU at higher cache storage environment. Our algorithm also generates less files I/O overhead at the cost of higher complexity and structure overhead.

We believe that this technique is suitable for web caching on mobile devices especially on low-end smartphones where RAM is severely limit but CPU power is sufficient.

References

1. [PODLIPNIG, 2003] Podlipnig, S., Böszörmenyi, L., A Survey of Web Cache Replacement Strategies, ACM Computing Surveys, Vol.35, No.4, December 2003, pp.374-398.
2. [MOOKERJEE, 2002] Mookerjee, V.S., Tan, Y., Analysis of a Least Recently Used Cache Management Policy for Web Browsers, Operations Research, Linthicum 50(2), 2002, pp.345-357.
3. [WILLIAMS, 1996] Williams, S., Abrams, M., Standridge, C.R., Abdulla, G., and Fox, E.A., Removal policies in network caches for World-Wide Web documents. In

proceedings of ACM SIGCOMM, ACM Press, 1996, pp. 293-305.

4. [WONG, 2006] Wong, K.Y., Web Cache Replacement Policies: A Pragmatic Approach, IEEE Network, 2006, pp. 28-34.
5. [LUTONEN, 1994] Luotonen, A., Altis, K., World-Wide Web Proxies, Computer Networks. And ISDN System, Volume 27, Issue 2, 1994, pp. 147-154.
6. [Romano, 2008] Romano, S., ElAarag, A quantitative study of recency and frequency based web cache replacement strategies, In proceedings of the 11th communications and networking simulation symposium, ACM New York, 2008.
7. [TIRDAD, 2009] Tirdad, K., Pakzad, F., Abhari, A., Cache replacement solutions by evolutionary computing technique, In proceedings of the 2009 Spring Simulation Multiconference, Society for Computer Simulation International, 2009.
8. [Ali, 2009] Ali, W., Shamsuddin, S.M., Intelligent Client-Side Web Caching Scheme Based on Least Recently Used Algorithm and Neuro-Fuzzy System, In Proceedings of the 6th International Symposium on Neural Networks: Advances in Neural Networks - Part II, Springer-Verlag Berlin Heidelberg, 2009.
9. [Torkzaban, 2009] Torkzabah, V., Rahmani, S., SCRAME: Selection of Cache Replacement Algorithm based on Multi Expert, In proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services, ACM New York, 2009.
10. [Geetha, 2009] Geetha, K., Gounden, N.A., Monikandan, S., SEMALRU: An Implementation of modified web cache replacement algorithm, World Congress on Nature & Biologically Inspired Computing, 2009.
11. [SRIVASTAVA, 2000] Srivastava, J., Cooley, R., Deshpande M., Tan, P., Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data. ACM SIGKDD Explorations Newsletter, Volume 1, Issue 2, January, 2000.
12. [Spiliopoulou, 2000] Spiliopoulou, M., Web usage mining for Web site evaluation. Communications of the ACM, Volume 43 Issue 8, 2000.
13. [Zhou, 2005] Zhou, B., Hui, S.C., Fong, A.C.M., Discovering and Visualizing Temporal-Based Web Access Behavior. In proceedings of the Web Inteligence 2005, IEEE Computer Society Washington, 2005.
14. http://ita.ee.lbl.gov/
15. http://www.web-caching.com