Web Browser Cache Management Techniques for Mobile Full Browsing on NAND-based Mobile Devices

Keonsoo Ha
School of Computer Science and Engineering
Seoul National University
air21c@davinci.snu.ac.kr

Jihong Kim
School of Computer Science and Engineering
Seoul National University
jihong@davinci.snu.ac.kr

ABSTRACT

As wireless communication and mobile device technologies improve, mobile full browsing is emerging as a killer application for mobile devices. In order to use mobile full browsing efficiently in terms of performance, energy, and download cost, mobile Web browsers adopt Web browser cache. However, the I/O access patterns of a Web browser cache make it difficult for these mobile devices to use a NAND-based secondary storage efficiently. In this paper, we propose Web browser cache management techniques for mobile full browsing on NAND-based mobile devices. A proposed filtering technique reduces useless write operations in NAND flash memory by selectively caching frequently reused Web pages only. A proposed NAND-aware packaging technique reduces garbage collection overheads by combining small files into a single NAND page. Our experimental results show that the proposed Web browser cache management techniques reduce the total response time of a Web browser cache on average 28% over a conventional Web browser cache management technique.

1. INTRODUCTION

Mobile full browsing, which allows users to access complete[1] Web pages from mobile devices, has not been widely used because many users find it unpleasant and cumbersome to navigate Web pages using a mobile Web browser. For example, in order to navigate complex Web pages from a typical cellular phone with a small display, a user must scroll up and down many times with frequent visits to hyperlinks for multiple subpages. Although manual and automatic reauthoring techniques [1, 2] provided a partial solution for this problem, they were not widely adopted because manual reauthoring severely limits the number of Web pages accessible from mobile devices while automatic reauthoring does not produce high-quality transcoded Web pages.

However, as faster mobile communication technologies, such as HSDPA [3] and WCDMA [4], are widely available and the computing power of mobile devices improves, mobile full browsing (combined with new interface techniques) is emerging as a killer application for mobile devices. For example, Apple's iPhone [5] has been successful by supporting mobile full browsing efficiently using a touch screen technology. Other cellular phone manufacturers have also introduced new cellular phones with a support for mobile full browsing. For example, LG's LH2300, Samsung's SPH-M800, and Nokia's N73 are available in the market. One of the common architectural characteristics of these new phones is that they all employ NAND flash memory as a secondary storage in addition to a relatively small main memory. In this paper, we assume that our target mobile devices are architected with a similar memory hierarchy consisting of two levels, the first-level (i.e., L1) main memory and the second-level (i.e., L2) NAND flash memory.

In order for these mobile devices to support full browsing efficiently, a mobile Web browser must use a Web browser cache. As commonly used for a similar purpose in a desktop Web browser, a Web browser cache can improve the Web access performance and reduce the energy consumption of the mobile devices during Web surfing. Furthermore, it can reduce the download cost for Web pages. For our target mobile devices with the two-level memory hierarchy structure, a Web browser cache is typically implemented in two levels. L1 Web browser cache is implemented in the main memory while L2 Web browser cache is implemented in the NAND flash memory. The two-level hierarchical structure of a Web browser cache takes advantages of both the main memory and the NAND flash memory. Although the main memory is faster than the NAND flash memory, its size is limited in mobile devices. By storing less frequently used Web pages in the NAND flash memory, we can have an effectively large and fast Web browser cache.

[1]For example, ones accessible from a typical desktop PC.

The two-level Web browser cache organization for our target mobile devices, however, requires a different cache management strategy from ones used for a desktop Web browser. For example, in a desktop PC, once downloaded into a local PC, a Web page is cached into a local Web browser cache. Although this strategy of caching all the downloaded files into a Web browser cache is reasonable for a desktop PC, it incurs a significant performance overhead for NAND-based mobile devices because of unique characteristics of NAND flash memory. First, since overwrite operations are not allowed in NAND flash memory, caching all the downloaded Web pages into a Web browser cache can cause unnecessary block copy operations during garbage collections. These extra data copy operations can significantly degrade the performance of Web browser cache, especially when cached Web pages are evicted from the Web browser cache without being reused. According to our observations, about 42% of cached Web pages are evicted without being reused. (Note that in a desktop PC with a hard disk, these unused files are not a problem because a hard disk can overwrite.) Therefore, it is necessary to filter out those downloaded files which are less likely to be reused.

Second, since a large number of files in the Web browser cache are smaller than the page size of NAND flash memory, which is 4 KB, garbage collections in NAND flash memory are frequently triggered. The garbage collection is triggered, for example, when the number of free blocks is below a preset determined threshold. In a log-based FTL, if the empty space of log blocks is exhausted, the garbage collection is triggered. Even if a file is smaller than the page size, one complete page is allocated to the file. Because of large internal fragmentations, the flash space is inefficiently utilized, triggering garbage collections frequently.

In order to overcome the mismatch problems between the NAND flash memory and the Web browser cache, we propose a new cache management strategy for a mobile Web browser cache based on NAND flash memory. First, we propose a filtering technique to reduce useless write operations into NAND flash memory which implements L2 Web browser cache. This technique prevents files which are less likely reused from being written into NAND flash memory. To decide if a file will be written to NAND flash memory or not, our proposed technique considers each file's history in L1 Web browser cache. Second, we propose a packaging technique to better handle a large number of small files in L2 Web browser cache. Our proposed NAND-aware packaging technique combines small files into a new file whose size is close to the page size of NAND flash memory so that the internal fragmentation problem can be alleviated. Experimental results based on several Web browser cache traces show that the proposed techniques reduce the total response time of a Web browser cache on average 28% over a conventional Web browser cache organization. In particular, the total response time in L2 Web browser cache which includes the garbage collection overhead is reduced on average 79%.

The rest of this paper is organized as follows. Section 2 summarizes related works and Section 3 introduces motivations. Section 4 describes the details of the proposed techniques. Experimental results are presented in Section 5. Finally, conclusions are given in Section 6.

2. RELATED WORKS

Web cache management has been widely studied for several years with a particular emphasis on replacement policies. GDS (Greedy Dual-Size) [6], GDSP (GDS Popularity) [7], and LNC-R-W3-U (Least Normalized Cost Replacement for the Web with Updates) [8] are the examples of better known replacement policies. In these algorithms, the main design goal is to enhance the performance of a Web proxy cache, thus the performance of a memory hierarchy in a server is not of interest. On the other hand, in our work, the main design goal is to enhance the performance of a Web browser cache of a mobile device. Therefore, we consider the performance of a memory hierarchy as a main design requirement.

Compared to Web cache techniques in servers, there are not many researches on mobile Web browser cache techniques. Yang *et al.* proposed a replacement algorithm in a mobile Web browser cache with memory and network bandwidth constraints [9]. They combined their replacement algorithm with partial caching and replacement. Jin *et al.* proposed Web caching and prefetching algorithms in mobile devices using sequence-mining based prediction algorithms [10]. In order to combine the caching replacement and prefetching algorithms together, they formulated the caching profit using the locality of a Web page, the size of a Web page, delay of saving a Web page. Like the studies on Web proxy server cache, the objective of the researches on mobile Web browser cache is to enhance the performance of the Web browser cache by better replacement algorithms and prefetching algorithms. However, like Web proxy cache studies, the performance of a secondary storage was not considered in existing mobile Web browser cache investigations. On the other hand, in our work, we enhance the performance of a Web browser cache in mobile devices by considering the characteristics of a secondary storage in mobile devices.

3. BASIC IDEA

3.1 NAND Flash Memory

NAND flash memory is organized with blocks and each block is composed of multiple pages. Each page is a unit of read and write operation, and each block is a unit of erase operation. Each block has the endurance limitation in terms of the number of erased operations. For example, one recent NAND flash memory consisted of MLC chips can be erased at most 10,000 times.

Unlike a hard disk, NAND flash memory does not allow overwrite operations. When the data in a page is updated, the new data is written to a free page and the old data must be invalidated. Because of this unique characteristic of NAND flash memory, a garbage collection scheme is needed to reclaim the invalidated pages. A garbage collection scheme should select a block which has many invalid pages and erase the selected block to be reused after copying the valid pages in the block to a new free block. Since a garbage collection requires many read, write, and erase operations, frequent invocations of a garbage collector can significantly degrade the performance of NAND flash memory.

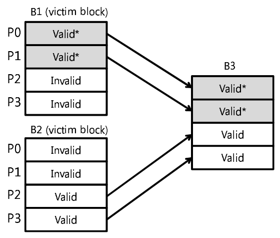

Figure 1: An example of extra operations during a garbage collection in L2 Web browser cache

3.2 Motivations

In this section, we explain the main motivations of our work using simple examples. Consider a two-level Web browser cache organization that consists of L1 Web browser cache and L2 Web browser cache, which were implemented in the main memory and NAND flash memory, respectively. Because NAND flash memory cannot support overwrite operations, if infrequently reused files are cached into L2 Web browser cache, they may cause a large number of useless copy operations during a garbage collection.

Figure 1 shows an example of such extra block copy operations during a garbage collection. There are three NAND flash memory blocks B1, B2, and B3. Each block consists of four pages. Grey colored pages store valid data but they are not reused. That is, copying the grey pages is useless for future computations, thus wasting CPU cycles. In Figure 1, we denote such valid pages using 'valid*'. Suppose that blocks B1 and B2 are the victim blocks to be erased during garbage collections. Before erasing the victim blocks, valid data in the victim blocks are copied into a new free block (block B3 is the free block in the figure). In this case, although data in pages P0 and P1 of the block B1 are not reused, two read operations and two write operations are necessary for a copying B1 to B3. In fact, if we had known it a priori that P0 and P1 of the block B1 are not reused, we could have avoided writing data to P0 and P1. Therefore, two read operations and four write operations were wasted.

If a large amount of data are not reused (e.g., P0 and P1 of B1), the performance of a Web browser cache can be deteriorated. Table 1 shows how many files are not reused after they have been saved in a Web browser cache. Three traces were collected from Web browser opera [11] while Web surfing. (For more details on there traces, refer Section 5.) Unused requests, which are used just once before they are cached to the Web browser cache, take on average 42 % of all the Web pages requests in our traces, assuming that the Web browser does not have any size limit. The statistics in Table 1 strongly suggest that we need an intelligent Web caching policy, possibly filtering out unused files from the L2 Web browser cache.

To understand the influence of extra operations during garbage collections on the performance of a Web browser cache, we observed breakdowns of the times spent by garbage collections in L2 Web browser cache using three traces. As shown in Figure 2, about

Table 1: Amount of Unused Requests in Web Cache Workloads

Trace	1	2	3
Total Requests (MB)	152	210	168
Unused Requests (MB)	70	81	68
Percentage of unused requests (%)	46	39	40

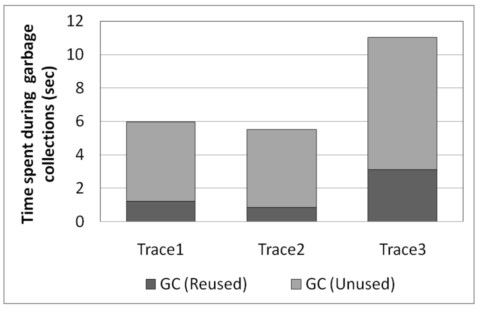

Figure 2: The breakdown of times spent during garbage collections of each trace

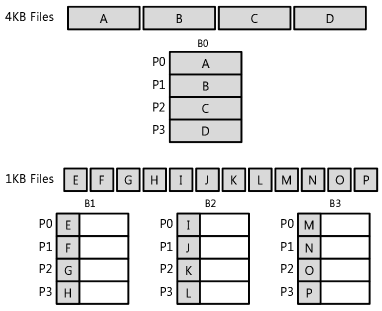

Figure 3: An example of internal fragmentations in L2 Web browser cache

79% of the total garbage collection time are wasted by useless copy operations.

The second motivation is based on our observations of downloaded file sizes. According to our observations for three traces, on average 95% of requested files are smaller than the page size. Even though a file is smaller than the page size, it takes a complete page, thus incurring an internal fragmentation. Since this phenomenon decreases the number of free blocks or reduces empty space of log blocks dramatically, garbage collections are triggered more frequently. Figure 3 shows an example of the internal fragmentation problem. There are four blocks and 16 files. The size of the files A, B, C, and D is 4 KB while the size of the rest of the files is 1 KB. In case of 4 KB files, each file occupies one page, taking one block B0. On the other hand, although the total size of 1 KB files is smaller than 16 KB, storing all 1 KB files requires three blocks. Since a garbage collection module is more frequently launched as there are more files accessed, a large number of small files will degrade the performance of the NAND flash memory. Therefore, a new technique to reduce the number of internal fragmentations in L2 Web browser cache is required.

4. WEB BROWSER CACHE MANAGEMENT TECHNIQUES

4.1 Overview of Web Browser Cache System

Our target mobile network system, which consists of Web servers, the internet, AP (Access point)s, and mobile devices, is presented in figure 4. Web servers store Web pages and the internet is a network of networks which consists of Web servers. APs are charge of connecting between mobile devices and the internet. A mobile device consists of a network interface, a processor, L1 Web browser cache, and L2 Web browser cache.

Figure 5 shows the operations in the Web browser cache system which we assume in our work. When a user visits a Web page with an URL (Uniform Resource Locator) address, a Web browser

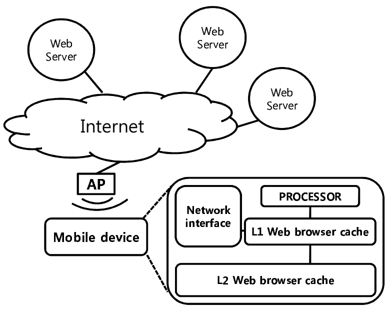

Figure 4: A mobile system environment

finds files related to the requested Web page using a Web browser database which enables a Web browser to find the requested file. If a file exists in a Web browser cache and it is valid, a Web browser obtains the file from a Web browser cache. A validation test is done by compared to the file in a Web server using the last modified time. In other cases, a Web browser downloads the file from Web servers.

When a file is stored in a Web browser cache, it is written into L1 Web browser cache. If the file is larger than the size of L1 Web browser cache, it is written to L2 Web browser cache directly. In the same manner, if a file is larger than the size of L2 Web browser cache, it is not cached in a Web browser cache. After the storing a file in a Web browser cache, the file information such as the file name and the URL address are updated in the Web browser database. If a file in L2 Web browser cache is updated, the file in L2 Web browser cache is invalidated because NAND flash memory cannot support overwrite operations. The updated file is stored in L1 Web browser cache.

If free space of L1 Web browser cache is not enough to cache a requested file, cached other files in the L1 Web browser cache are evicted by a replacement policy to make free space. In our work, we use LRU (Least Recently Used) as a replacement policy. The evicted files from the L1 Web browser cache are stored in L2 Web browser cache. In the same way, if free space in L2 Web browser cache is insufficient, same processes are done. The evicted files from the L2 Web browser cache are removed and the information about the evicted file in a Web browser database is also deleted.

To utilize fast access time of the main memory efficiently, frequently accessed files in L2 Web browser cache can be migrated to L1 Web browser cache. Several files in L1 Web browser cache must be evicted to make free space for the migrated file. Moreover, some files in L2 Web browser cache may be evicted to make free space for storing the evicted files. Thus, frequent migration can degrade performance of a Web browser cache. To prevent a migration from being triggered frequently, we assume that a frequently accessed file is migrated when a migration profit is larger than the cost. The cost can be denoted as $\left\lceil \frac{f.size}{N.pagesize} \right\rceil * N_W$,

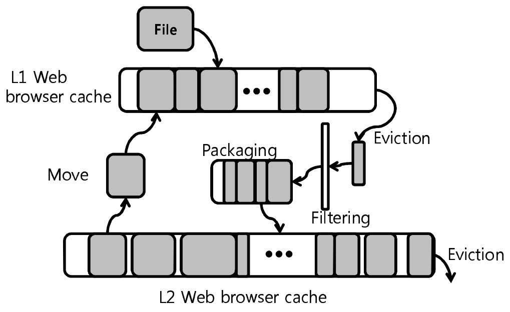

Figure 5: The operations in the Web browser cache

where $f.size$, $N.pagesize$, and N_W are size of a migrated file, size of a NAND flash memory page, and latency time of a write operation in NAND flash memory, respectively. This is because that the evicted files may be written to L2 Web browser cache taking as much as the space which the migrated file has occupied. We assume a migration profit as $\left\lceil \frac{f.size}{N.pagesize} \right\rceil * file_{accessInNAND} * N_R$, where $file_{accessInNAND}$ and N_R are the number of accesses of a file in L2 Web browser cache and latency time of a read operation in NAND flash memory, respectively. (Access time to be taken in main memory is not considered in this profit, because it is negligible because of its fast bandwidth.) If a migrated file is accessed as many as $file_{accessInNAND}$ times in L1 Web browser cache after a migration, the amount of the profit can be saved in terms of time.

Our proposed filtering technique and packaging technique are applied when a file is evicted from L1 Web browser cache. The details of the techniques are described in Section 4.2 and Section 4.3.

4.2 Filtering Technique

Filtering technique is to write requested Web pages into NAND flash memory selectively to reduce useless copy operations. The basic idea of this technique is to prevent files, which have been accessed infrequently in L1 Web browser, from being written to L2 Web browser cache. Since users tend to visit the Web pages which they have visited habitually, more frequently accessed Web pages in recent has a high probability of being reused. However, although a Web page is frequently accessed in L1 Web browser cache, if it is updated frequently in L2 Web browser cache, performance of L2 Web browser cache can be reduced due to garbage collections. Thus, this technique also considers the number of updates of a file in L1 Web browser cache.

In order for this technique to filter out the files which are infrequently accessed and frequently updated, this technique uses two thresholds. One is ‘access threshold’ and another is ‘update threshold’. When a file is evicted from L1 Web browser cache, this algorithm compares the number of accesses and updates of the evicted file to the thresholds. If the number of accesses of the file is larger than the access threshold and the number of updates of the file is less than the update threshold, it is written to L2 Web browser cache. In other cases, the evicted file is removed and the information about the file in a Web browser database is removed.

The thresholds must be changed to reflect users’ Web page access patterns and Web page characteristics. If users’ Web page accesses are centralized into a small number of Web pages, the files related to the Web pages are accessed more frequently. In this case, a high access threshold is preferred, because it can avoid rarely accessed files being cached into L2 Web browser cache. On the other hand, if Web pages are visited evenly among all, a low access threshold is preferred. Since tight filtering deletes most files, a Web browser cannot obtain many files from Web browser cache. In the same manner, if Web pages which users have visited are updated frequently, a low update threshold is preferred to prevent them from being stored in L2 Web browser cache. On the opposite case, a high update threshold is preferred to avoid excessively tight filtering.

Algorithm 1 Threshold changing algorithm

```
1: if $(Miss_a + Miss_u) = scope$ then
2:     if ( $Miss_a > PrevMiss_a) \vee (Miss_a = scope)$ then
3:         $th_a \leftarrow th_a + 1$
4:     else
5:         $th_a \leftarrow th_a - 1$
6:     end if
7:     if $(Miss_u > PrevMiss_u) \vee (Miss_u = scope)$ then
8:         $th_u \leftarrow th_u - 1$
9:     else
10:        $th_u \leftarrow th_u + 1$
11:    end if
12:    $PrevMiss_a \leftarrow Miss_a$
13:    $PrevMiss_u \leftarrow Miss_u$
14:    $Miss_a \leftarrow 0$
15:    $Miss_u \leftarrow 0$
16: end if
```

In our proposed technique, Threshold changing algorithm is charge of changing each threshold value dynamically.The basic idea of this algorithm is to change the thresholds by evaluating current threshold values. The proposed filtering technique predicts the probabilities of both being reused and being updated of an evicted file using current access threshold and update threshold. If current thresholds are not proper for current Web page access patterns, the number of misjudgments increases. For example, if an access threshold was quite low, many files are evicted from L2 Web browser cache without being reused. It implies that the predictions based on the access threshold were wrong. In our paper, we call the wrong prediction by the access threshold ‘access miss-prediction’. In the same manner, if an update threshold was quite high, many files are invalidated by being updated in L2 Web browser cache. It implies that the predictions based on the update threshold were wrong. We call the wrong prediction by the update threshold ‘update miss-prediction’. Therefore, the variances of the number of access miss-predictions and update miss-predictions can be important hints to evaluate current thresholds.

Changing threshold algorithm changes the thresholds using the variances of the number of access miss-predictions and update miss-predictions. Algorithm 1 shows how this algorithm works. The access miss-prediction, denoted as $Miss_a$, increases whenever a file, which have not been reused, is evicted from L2 Web browser cache. The update miss-prediction, denoted as $Miss_u$, increases whenever a file is updated in L2 Web browser cache. When the summation of access miss-predictions and update miss-predictions is equal to a pre-defined value $scope$, this algorithm changes the thresholds. (We set the $scope$ with 20 based on several experiments.) This algorithm checks whether the number of miss-predictions increases or not compared to the last evaluation. If $Miss_a$ is larger than $PrevMiss_a$, which is the number of access miss-predictions in the previous evaluation, the access threshold th_a increases. Since current access threshold is so low, the number of access miss-predictions increases compared to the last evaluation. Thus, the access threshold increases for tighter filtering. Moreover, if $Miss_a$ is equal to $scope$, the access threshold increases for the same reason. In the opposite case, the access threshold decreases for looser filtering. If the access threshold is equal to zero, it does not decrease. In the same manner, if $Miss_u$ is larger than $PrevMiss_u$, which is the number of update miss-predictions in the previous evaluation, the update threshold th_u decreases. Since current update threshold is so high, the number of update miss-predictions increases compared to the last evaluation. Therefore, the update threshold decreases for tight filtering. If the update threshold is equal to zero, it does not decrease. Moreover, if $Miss_u$ is equal to $scope$, the update threshold decreases for the same reason. In the opposite case, the update threshold increases for looser filtering. After the changing thresholds, $Miss_a$ and $Miss_u$ are copied to the $PrevMiss_a$ and $PrevMiss_u$, respectively. They are used in the next evaluation. Finally, $Miss_a$ and $Miss_u$ are initialized with zero.

4.3 Packaging technique

Packaging technique is to combine files smaller than the page size of NAND flash memory into a file of which size is close to the page size. When a file is evicted from L1 Web browser cache, this technique compares the size of the evicted file to the page size. If the evicted file is smaller than the page size, a file is created to store small files. For convenience of describing, we call the created file for packaging ‘package file’. Since the evicted file makes an internal fragmentation, it is copied to a package file instead of being written to L2 Web browser cache. In order to utilize space in a page of NAND flash memory as efficiently as possible, this technique collects other small files to be packaged by scanning L1 Web browser cache. The order of scanning follows the order of replacement in L1 Web browser cache. If a file, which is so small that the summation of the file size and the size of the package file is smaller than the page size, is found, this technique makes a replication of the file and copies it to the package file. A file, which has been replicated, does not be considered again when other package files scan for packaging. This technique does not change the order of replacement in L1 Web browser cache, because it uses a replication instead of evicting a file. These processes are ended if the size of a package file becomes same to NAND flash memory size or all files in L1 Web browser cache are scanned. After the scanning, the package file is written to L2 Web browser cache. Like a normal file, the information of the package file is added to a Web browser database when the package file is written to L2 Web browser cache.

Figure 6 presents an example of this technique. Assume that file B is selected as a victim file by a replacement policy in L1 Web browser cache. This file is copied to a package file, because it is smaller than a page of NAND flash memory. This algorithm scans L1 Web browser cache and the replications of files C and D are copied to the package file. The file between the files C and D are not selected because of its large size. A dotted line and small letters in the figure denote copying a replication and replications, respectively. In the example in Figure 3, four 1KB files can be packaged into a 4 KB file. Therefore, all 1 KB files can be stored in a block.

To read a packaged file in a package file, a Web browser database in a Web browser cache keeps the information about packaging. When a file is packaged in a package file, information in a Web browser database is updated. For a packaged file, the identification of the package file and an offset in the package file are updated. If a packaged file is requested, Web browser finds the file using the identification and the offset in a Web browser database. On the other hand, for a package file, the number of packaged files and total size of the packaged files are stored. When a file is packaged in a package file, the number of packaged files and the size of the packaged file increase. If a packaged file in a package file is invalidated by being updated, the number of packaged files decreases one and size of a package file decreases as the size of the invalidated file. If the number of included files in a package file becomes zero, the package file is removed from L2 Web browser cache.

To utilize fast access time of L1 Web browser cache, if a packaged file is accessed frequently in L2 Web browser, the replication of the file is created and it is copied to L1 Web browser. The condition for a migration triggering is same to the condition for normal files, but this technique makes a replication instead of moving the original file. This is why that a write operation is saved when the replication is evicted from L1 Web browser cache if the original file of the replication exists in L2 Web browser cache.

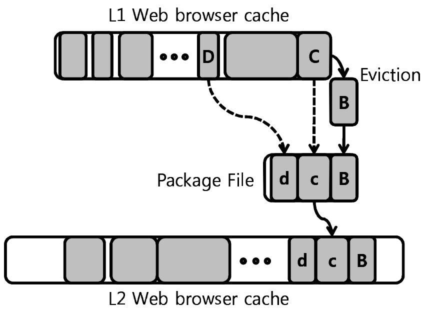

Figure 6: An example of the proposed packaging technique

Table 2: Characteristics of Traces

Trace	1	2	3
The number of files	3,980	3,966	4,299
Write request (MB)	85	102	101
Read request (MB)	67	108	67
Percentage of under 4KB (%)	98	98	89
Byte hit ratio (%)	44	51	40

Table 3: System Configuration Parameters

Main memory (L1 Web browser cache)	Size of Web caching : 2 MB Access Latency : 127.5 ns
NAND flash memory (L2 Web browser cache)	Size of Web caching : 15 MB Capacity : 512 MB Read latency : 60 us Write latency : 800 us Erase latency : 1.2 ms Page size : 4 KB Block size : 512 KB
Network bandwidth	14.4 Mbps

5. EXPERIMENTS

In order to evaluate the proposed techniques, we have developed a trace-driven Web caching simulator as well as a NAND flash memory simulator. In order to collect I/O access traces of the Web browser cache, we used the I/O access trace program in AccuSim [12]. Since the program captures all I/O traces from a Web browser, we extracted the I/O traces of Web browser cache from Web browser I/O access traces. We have collected the traces from a desktop PC without a specific scenario to mimic usual full browsing behaviors such as sending an e-mail, visiting portal sites, and Web searching. Table 2 shows information of each trace such as the number of requested files, the amount of read request, the amount of write request, a percentage of requested file which is smaller than 4KB, and byte hit ratio. The byte hit ratio is obtained assuming an infinite size of a Web browser cache. Table 3 summarizes system configuration parameters used for our experiments. We assume that users set the Web browser cache size as 2 MB for L1 Web browser cache and 15 MB of NAND flash memory for L2 Web browser cache in 512 MB flash memory. The access latency of the main memory is calculated based on [13]. We use the log-based FTL [14] for evaluations and set the number of log blocks as 10.

Figure 7 shows the total response times in L2 Web browser for each trace under three different management schemes. ‘None’, ‘F’, ‘P’, and ‘FP’ in the x-axis represent a conventional Web browser, the filtering technique, the packaging technique, and the combination of the filtering technique and packaging technique, respectively. ‘None’ is the base case for comparisons. The y-axis indicates the total response time in L2 Web browser cache. The proposed filtering technique and the proposed packaging technique reduce the total response times on average 72% and 38% over the base case. Finally, the combined technique reduces the total response time about on average 79% over the base case. This implies that the proposed techniques improve the performance of L2 Web browser cache by selective writing and deferring garbage collection triggerings.

Figure 8 shows the breakdown of the total response times of the Web browser cache. Since the network latency is very large and about 55% of Web pages are downloaded through Web servers,

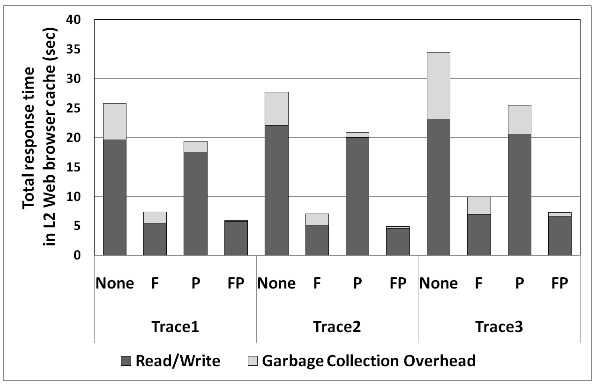

Figure 7: The breakdown of total response times in L2 Web browser cache

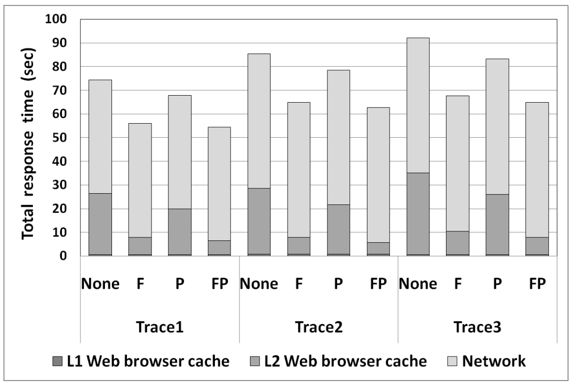

Figure 8: The breakdown of total response times of Web browser cache

it has the largest portion in the breakdown. On the other hand, although many files are obtained directly from L1 Web browser cache, it has the smallest portion in the breakdown because of its fast access time. The filtering technique and the packaging technique reduce the total response times of the Web browser cache on average about 25% and 9% comparing to the base case, respectively. The combined technique reduces the total response times about 28% on average. Moreover, we observed that the byte hit ratios of each case are almost similar. It implies that the proposed techniques can enhance the performance of a Web browser cache by reducing overheads of a secondary storage.

6. CONCLUSIONS

We have proposed two new Web browser cache management techniques appropriate for mobile devices that support mobile full browsing. Taking account of NAND flash memory's unique characteristics, the proposed filtering technique effectively distinguishes files with high reusability from ones with low reusability. By saving files with high reusability only into L2 Web browser cache in NAND flash memory, the proposed filtering technique significantly reduces the garbage collection overhead of NAND flash memory over a conventional technique.

The proposed packaging technique minimizes internal fragmentations which frequently occur when the size of downloaded files is smaller than the page size of NAND flash memory. By intelligently combining small files into larger files that fit within the page boundary without introducing internal fragmentations, the proposed packaging technique also reduces unnecessary garbage collections. Experimental results demonstrate that the proposed techniques can reduce the total response time of a Web browser cache on average 28% over a conventional approach.

Our current work can be extended in several directions. For example, as an immediate future work, we plan to implement the proposed techniques in one of the open-source mobile phones such as Openmoko's Neo FreeRunner phone [15] to evaluate the performance benefits of the proposed techniques in a practical setting. Moreover, we will extend our experiments with various FTLs. Since the condition for a garbage collection triggering and the processes in a garbage collection depend on a FTL, considering characteristics of FTLs in Web browser cache will be an interesting issue.

7. ACKNOWLEDGMENTS

This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the National Research Lab. Program funded by the Ministry of Education, Science and Technology (No.R0A-2007-000-20116-0). This work was also supported in part by the Brain Korea 21 Project in 2008. The ICT at Seoul National University provides research facilities for this study.

8. REFERENCES

[1] K. Nagao, Y. Shirai, and K. Squire. "Semantic annotation and transcoding: making Web content more accessible," IEEE Multimedia, vol. 8, no. 2, pp. 69-81, 2001.

[2] J. Chen, B. Zhou, J. Shi, and H. Zhang. "Function-based object model towards website adaptation," in Proc. International Conference on World Wide Web, 2001.

[3] http://hspa.gsmworld.com

[4] http://www.umtsworld.com/technology/wcdma.htm

[5] http://www.apple.com/iphone

[6] P. Cao and S. Irani. "Cost-aware WWW proxy caching algorithms," in Proc. USENIX Symposium on Internet Technologies and Systems, 1997.

[7] S. Jin and A. Bestavros. "Popularity-aware greedy dual-size Web proxy caching algorithms," in Proc. International Conference on Distributed Computing Systems, 2000.

[8] J. Shim, P. Scheuermann, and R. Vingralek. "Proxy cache algorithms: design, implementation, and performance," IEEE Transaction on Knowledge and Data Engineering, vol. 11, no. 4, pp. 549-562, 1999.

[9] C. Yang, K. Tien, and M. Wueng. "Browser cache management for small wireless devices with memory and bandwidth constraints," in Proc. Parallel and Distributed Computing, Applications and Technologies, 2003.

[10] B. Jin, S. Tian, C. Lin, X. Ren, and Y. Huang. "An integrated prefetching and caching scheme for mobile Web caching system," in Proc. International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007.

[11] www.opera.com

[12] A. R. Butt, C. Gniady, Y. C. Hu. "The performance impact of kernel prefetching on buffer cache replacement algorithms," ACM SIGMETRICS Performance Evaluation Review, vol. 33, no. 1, pp. 157-168, 2005.

[13] Samsung Electronics. "4M × 32Bit × 4 Banks Mobile SDRAM," http://www.samsung.com/global/system/business/semiconductor/product/2007/6/11/MobileSDRAM/MobileSDRSDRAM/512Mbit/K4M51323PC/ds_k4m51323pc.pdf, 2007.

[14] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho. "A space-efficient flash translation layer for compact flash systems," IEEE Transactions on Consumer Electronics, vol. 48, no. 2, pp. 366-375, 2002.

[15] http://wiki.openmoko.org