Reducing the User-perceived Latency of Browsers with NVRAM

Kyusik Kim, Yongwoon Cho, Seongmin Kim, and Taeseok Kim

Abstract—Non-volatile RAM (NVRAM) provides many opportunities to improve the performance of computing devices. In this paper, we present an approach that reduces the user-perceived latency of browsers by using NVRAM. To this end, we first analyze the browser launch process, and then employ several techniques that improve the performance of each step by using NVRAM. Specially, we focus on minimizing the launch time of browser by 1) prefetching the block sequence required for browser launch, 2) caching the web resources in the fast NVRAM, and 3) reusing the displayed bitmap data in the frame buffer. Through implementation, we show that our scheme significantly reduces the launch time of browsers.

Index Terms—NVRAM, fast launch, browser, frame buffer, caching

I. INTRODUCTION

With the fast advance of semiconductor technologies, research and development of NVRAM are being actively performed in academia and industries. The representative examples of NVRAM are STT-RAM, PCRAM, and ReRAM. Even though the physical characteristics of them are not identical, they are commonly fast, non-volatile, and byte addressable. The NVRAM is anticipated as the next generation storage media, and thus research domain is spreading in semiconductor devices, system architectures, operating systems, and application programs order [1-4].

How to effectively use the NVRAM in computing devices is still an open problem. This paper focuses on how to make use of the NVRAM for the browser performance improvement. To the best of our knowledge, this is the first paper that presents an approach using NVRAM for applications except for databases. The browser is one of the most popular applications in computing devices. According to the smartphone application usage statistics, the browser is still widely used in smartphones as well as PCs, occupying 10-12% of all applications [5]. It is originally an application that retrieves and then presents the web resources like text and image from web servers, but now it has been evolved into a platform with the appearance of HTML5. Many users enjoy various services such as SNS (Social Network Services) and games through the browsers, so the performance of browser becomes increasingly important.

In this paper, we present a novel approach to improve the user-perceived performance of browsers by making use of NVRAM. Since application launch time is an important metric for end-users, we specially focus on the launch time of browsers. In order to reduce the launch time of a browser, we first analyze the browser launch process and then employ some techniques that can improve the performance by using NVRAM. The first step of browser launch process is to load the browser executable, libraries, and configuration files from the storage. We can reduce the file loading time by extracting the files required for browser launch and placing them in the NVRAM faster than NAND flash memory. In addition, we obtain the block sequence
required for browser launch by repeating the browser execution and load them into the memory buffer in advance when execute the browser.

The second step of browser launch is to make some widgets and load the default web page configured by users. In order to launch the browser fast, we exceptionally cache the bitmap data in the frame buffer, which is displayed as a result of web page processing. When a user executes the browser, our scheme restores the cached bitmap data into the frame buffer again. Through this scheme, the user may feel that the browser is instantly launched. However, as it is just a bitmap data, the user cannot interact with the displayed web page. To solve this problem, we perform an actual web page processing as a background task simultaneously. When the background task finishes the actual web page processing, our scheme replaces the displayed bitmap data with it transparently. The actual web page processing time should be short enough for users not to feel this replacement. It is mainly affected by the web resource loading time, which depends on the network latency [6]. To reduce the resource loading time, a caching technique can be an effective solution; we cache the web resources in high speed NVRAM instead of NAND flash memory.

The remainder of this paper is organized as follows. Section II presents the related works. In Section III, the fast browser launch scheme using NVRAM is explained in detail with the analysis of browser launch process. The performance evaluation of the proposed scheme is described in Section IV. Finally, we conclude this paper in Section V.

II. RELATED WORKS

There have been several studies for fast application launch. In [7], they proposed a prefetching scheme that overlaps the computation time and I/O access time during an application launch. Actually, their work provided many hints for us to make a research. For implementation, they used a set of user-level components and system debugging tools provided by Linux. As a result, application launch time is reduced by 28% on average. A scheme that pre-fetches the required pages prior to page fault has been also proposed [8]. A scenario file comprising a list of pages that need to be pre-fetched is first created. A scenario that requires pre-fetching may be automatically detected when application launches. In [9], the authors proposed a method to predict and pre-execute an application for user to execute in the next. The application is selected by analyzing temporal access pattern.

There have been also some studies for improving the software start time by exploiting the byte addressability of NVRAM. In [10], the authors proposed a technique to shorten the booting time by using NVRAM. It eliminates kernel loading and decompression steps by maintaining the kernel image on the NVRAM. [3] presented a framework called HEAPO (Heap-Based Persistent Object Store) to manage persistent objects in the NVRAM. By exploiting the byte addressability and non-volatility of NVRAM, it defines its own persistent heap layout, persistent object, and name space structure; they can be used for software boosting.

The performance improvement of browsers was another hot research topic for a long time. The performance of browser depends on resource loading time, page rendering time, and the javascript execution speed [12]. Among them, slow resource loading time becomes a performance bottleneck in mobile devices, which are based on wireless network. In order to reduce the resource loading time, caching technique is widely used. Wang et al. analyzed a lot of real browsing histories from smartphones, and showed a study on the effectiveness of the mobile browser’s cache [6]. They first showed that over half of the web pages are not optimized for mobile devices and the browser is slow because of the network round-trip time. They also found that increasing the cache size of the browser on smartphones may not improve the performance of browsers and revalidating the cached resources greatly reduces the effectiveness of browser cache [13].

III. FAST LAUNCH OF BROWSERS USING NVRAM

The application launch time consists of a sequence of the I/O access times and the computation times. The first block required for application launch is read from the storage and then the CPU proceeds the launch process with the block. This process is repeated until the launch of an application is finished as can be seen in Fig. 1(a). It
is a cold start scenario, which is the first launch of an application since the system boot-up [7].

The launch of browser also starts by loading the files required for launch, for example an executable, libraries, and configurations from the storage. We can extract these files with `strace` tool, which is a profiler for tracing the system calls. During browser launch, we monitor only the system calls that have a file name as their argument like `open()`, `creat()`, `execve()`, and `stat()`. If we place these files required for browser launch into the NVRAM faster than NAND flash memory up to 2,500 times, the launch time of browser can be significantly reduced (Fig. 1(b)) [3].

Since the blocks required for browser launch are fixed, we can extract the block sequence required by executing the browser repeatedly. To track the block sequence, we use `blktrace` that is a built-in Linux kernel I/O tracing tool for monitoring the block I/O requests [14]. The block sequence extracted is saved as a file and read in advance by prefetcher when the browser launches. For the prefetcher to read the block sequence, the low-level block information like logical block address should be first translated to file system information such as file name, offset, and size because prefetcher is implemented as another application. To this end, we map block information to file system information by using debugfs, which is Ext file system debugger.

Finally, the prefetcher is implemented by using `posix_fadvise()`, which can read the blocks in advance before the browser actually requests them. When we execute the browser, the prefetcher is also performed simultaneously and the blocks for browser launch are read into the memory buffer. As a result, the browser launch time can be reduced as Fig. 1(c). If the browser launches when all blocks are already kept in the memory buffer, the launch time of browser can be reduced as Fig. 1(d). It is called warm start. The whole prefetching process for fast browser launching is illustrated in Fig. 2.

When loading of blocks required for browser launch is finished, the execution of browser starts by invoking main function. In this step, the browser makes some widgets such as bookmark, toolbar, and menu, and configures window attributes. The final step is to process the default web page, which is configured by user. With the default web page’s URL, the browser first fetches the web resources belong to the page from the remote web server or the local browser cache [12]. Then, it parses the HTML into a DOM tree, processes CSS for style formatting, executes javascript, etc. It attaches such intermediate objects to the internal representation (IR), and computes the screen locations based on the IR. Finally, the browser paints the IR by putting the final graphical bitmap data into the frame buffer.

For the fast launch of browser, we cache the bitmap data, which is displayed as a final result of web page processing. When users want to launch the browser, our scheme just restores the cached bitmap data into the frame buffer again. Since this scheme removes both the second step that makes some widgets and third step that processes the default web page, users may feel that the browser is instantly launched. However, since it is just a bitmap data, users cannot click links, write texts, play audio/video, etc. with the displayed web page. To address this problem, we should still perform the original web page processing task in the background. Almost
modern browsers perform the steps for web page processing such as resource loading, processing, attachment, and painting in a pipelining manner. Therefore, the window that processes the default web page in the background should be temporarily set to invisible because it may partly update the display and thus it may give some inconvenience to users. Fig. 3 illustrates the proposed frame buffer caching scheme.

When the background task finishes the web page processing, our scheme transparently replaces the displayed bitmap data with the actually processed data by setting visible. The actual page processing time should be short enough for users not to feel this replacement. To reduce the actual page processing time, we build a caching space in the NVRAM faster than the NAND flash memory. By caching the web resources of default page as well as the frame buffer bitmap data in the NVRAM, we can improve both the user-perceived browser launch time and the actual page processing time. Almost modern browsers employ the browser cache, which stores the web resources of the pages visited in the storage and then uses them to render when users revisit the pages. If we additionally build the browser cache in the NVRAM, the processing time of all web pages that users revisit will be also reduced.

IV. PERFORMANCE EVALUATION

To demonstrate the effectiveness of the proposed scheme, we developed our scheme by writing some scripts and modifying QT-webkit, which is an open source browser. Table 1 shows the summary of the embedded board used in our experiments. As the NVRAM is not available in our experiments, we used a DRAM based RAMDISK instead. Note that the I/O performance of NVRAM is similar to or a little lower than that of DRAM [3]. We selected several popular sites from the Alexa top 500 sites as the default web page [11].

With various default web pages, we first analyze the browser launch time into three: 1) $T_{\text{load}}$, the time to load the browser launch related files from the storage, 2) $T_{\text{widget}}$, the time to make widgets and configure window attributes, and 3) $T_{\text{page}}$, the time to process the default web page. Fig. 4 shows the analysis results. As expected, both $T_{\text{load}}$ and $T_{\text{widget}}$ show similar values regardless of the default web page setting; $T_{\text{load}}$ is about 2.2 s and $T_{\text{widget}}$ is about 2.1 s, respectively. However, the values of $T_{\text{page}}$ are different depending on default web pages because it is affected by network I/O performance and the components of the pages. Except for blank page, the browser launch time mainly depends on $T_{\text{page}}$, which is from 2.44 s to 14.67 s in our experiments.

Fig. 5 shows the performance of the proposed scheme. To show the effectiveness of the proposed scheme in detail, we measured the performances of no scheme, prefetching scheme, frame buffer caching (FBC) scheme, and prefetching and frame buffer caching scheme, respectively. As can be seen, the browser launch time is significantly reduced when the prefetching scheme is employed because there is little I/O access time. When the frame buffer caching is also employed, the browser launch time is significantly reduced because there is no

![Frame buffer caching scheme for reducing the user-perceived latency.](image)

![The analysis of browser launch time.](image)

<table>
<thead>
<tr>
<th>Table 1. Experimental Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
</tr>
<tr>
<td><strong>DRAM</strong></td>
</tr>
<tr>
<td><strong>Display resolution</strong></td>
</tr>
<tr>
<td><strong>Linux Kernel</strong></td>
</tr>
<tr>
<td><strong>GUI framework</strong></td>
</tr>
</tbody>
</table>
making widget time or default web page processing time. If prefetching and frame buffer caching are employed together, the browser launch time is about 0.35 seconds. It is very low because there is no extra operation except for restoring the cached bitmap data. Note that it is the very user-perceived browser launch time in our scheme and is constant irrespective of default web pages. In our scheme, the space for storing the bitmap data is additionally required, and we store it in the fast NVRAM. The size of bitmap data to be stored depends on the display resolution, and it is 3.2 MB in our experiment.

As users cannot interact with the bitmap data, the browser launch time that actually processes the default web page as a background task should be also short enough. In order to reduce this time, we build the browser cache in the NVRAM with fast data access speed. In Fig. 6, we can see that the default web page processing times are significantly reduced in our scheme. It is because the blocks of libraries and configurations for handling the web page already exist on the memory buffer because of prefetching as well as many web resources are restored from NVRAM based browser cache.

Fig. 5. The performance of the proposed scheme.

If the gap between the user-perceived launch time and the background actual launch time is over about 1-2 seconds, our scheme may be useless. In our experiments, if blank page or Google site is set as a default web page, our scheme can be very useful. However, if users want the Yahoo.com or Amazon.com site as a default web page, our scheme may be not practical. It is important to note that this experiment was performed with the low-end embedded board. We anticipate that our scheme will become more effective as the hardware technologies including CPU, network, and storage continuously improve.

V. CONCLUSIONS

In this paper, we presented a scheme that improves the browser launch time by using NVRAM. We could reduce the browser launch time significantly by prefetching of the block sequence required for browser launch and caching the web resources in the fast NVRAM. Specially, we could achieve an almost instant launch of browser by reusing the displayed bitmap data in the frame buffer. We believe that our scheme can be also applied in other applications without serious modification.

ACKNOWLEDGMENTS

This work is supported by IT R&D program MKE/KEIT (No. 10041608, Embedded System Software for New-memory based Smart Device). This research has been also conducted by the Research Grant of Kwangwoon University in 2014.

REFERENCES


**Kyusik Kim** received the BS degrees in computer engineering from Kwangwoon University, Korea, in 2014. He is currently working toward the combined MS and PhD degree at the School of Computer Engineering, Kwangwoon University. His research interests include next-generation nonvolatile memories, and embedded system.

**Yongwoon Cho** received the BS and MS degrees in computer engineering from Kwangwoon University, Korea, in 2012 and 2014, respectively. He is currently working toward the PhD degree at the School of Computer Engineering, Kwangwoon University. His research interests include operating systems, flash memories, web browser and embedded system.

**Seongmin Kim** received the BS and MS degrees in computer engineering from Kwangwoon University, Korea, in 2011 and 2014, respectively. He is currently working toward the PhD degree at the School of Computer Engineering, Kwangwoon University. His research interests include operating systems, flash memories and next-generation nonvolatile memories, and embedded system.

**Taeseok Kim** received the BS, MS and PhD degrees in computer science from Seoul National University, Korea, in 2000, 2002, and 2007 respectively. He is currently an associate professor in the department of computer engineering, Kwangwoon University, Seoul, Korea. His research interests include multimedia systems, operating systems, storage systems, and embedded systems.