Similarity-based Web Browser Optimization

Haoyu Wang, Mengxin Liu, Yao Guo*, Xiangqun Chen
Key Laboratory of High-Confidence Software Technologies (Ministry of Education)
School of Electronics Engineering and Computer Science,Peking University, Beijing, China
{wanghy11,liumx11,yaoguo,cherry}@sei.pku.edu.cn

ABSTRACT

The performance of web browsers has become a major bottleneck when dealing with complex webpages. Many calculation redundancies exist when processing similar webpages, thus it is possible to cache and reuse previously calculated intermediate results to improve web browser performance significantly. In this paper, we propose a similarity-based optimization approach to improve webpage processing performance of web browsers. Through caching and reusing of style properties calculated previously, we are able to eliminate the redundancies caused by processing similar webpages from the same website. We propose a tree-structured architecture to store style properties to facilitate efficient caching and reuse. Experiments on webpages of various websites show that the proposed technique can speed up the webpage loading process by up to 68% and reduce the redundant style calculations by up to 77% for the first visit to a webpage with almost negligible overhead.

Categories and Subject Descriptors

H.4.3 [**Communications Applications**]: Information Browsers

Keywords

Web Browser; Similarity; Caching; Cascading Style Sheet

1. INTRODUCTION

Web browser has become one of the most important applications on a variety of computing devices, including PCs, tablets and mobile phones. A majority of users rely heavily on web browsers to access information, data and services on the Internet. However, as webpages become more and more complex, the performance of browsers has become critical for users to have a satisfactory browsing experience. Due to the limitation of network bandwidth and processing power, loading and processing complex webpages might take several seconds to tens of seconds. This performance bottleneck of web browsers is especially worse on resource-constrained mobile devices such as smartphones.

*corresponding author

WWW'14, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2744-2/14/04.
http://dx.doi.org/10.1145/2566486.2567971.

The webpage loading time is affected by two major factors. The first factor is due to the time cost when retrieving the web contents from a remote web server, which is limited by the networking speed. The other factor is the time of processing the webpage locally. As webpage processing is computation-intensive and needs substantial computation resources, it has become a very important factor affecting the performance of web browsers, especially for the mobile devices with weak computing power.

Because networking speed is typically out of the control of web browsers, in this paper we mainly focus on improving the webpage processing speed on the devices. According to previous research [9, 29] and our own experimental results, style formatting and layout calculation account for most of the webpage processing time and they are the key issues that lead to poor web browsing performance.

With a close inspection of the calculations performed during webpage processing, we find out that there exist many redundancies that can be potentially eliminated, mainly due to similarities between webpages and the embedding components. For example, many webpages might be visited more than once. Although related resources of these webpages are typically cached, the browser still need to calculate the styles and layouts every time the webpage is accessed. On the other hand, multiple webpages from the same website also exhibit strong similarities, causing the browser performing many repetitive calculations. Caching and reusing these intermediate calculated results could potentially improve the performance of web browsers, as demonstrated by earlier studies such as Smart Caching [29].

In this paper, we explore the similarities between webpages and propose a new approach to eliminate the redundant style calculations between webpages by caching the style formatting results and sharing the results between different webpages. The key technique in our proposed approach is *similarity-based style reuse*. We store the processed style properties in tree-structured caches which can be shared between webpages. Cached style properties could be loaded directly without re-calculation when the identical element is processed subsequently.

Our work is partially motivated by Smart Caching [29], which first proposed to reduce the local computations by caching the intermediate results when revisiting the same webpage. The Smart Caching approach is effective for webpage revisits, however, the majority of the webpage visits are new visits. For example, 75% of the webpages visited are new visits for mobile user in a recent study [24]. Besides same-page revisits, we explore the possibilities of exploiting the similarities between webpages from the same website, analyze the amount of similar styles between webpages and propose a new technique supporting style reuse between different webpages.

Compared to caching techniques for the same-page revisits, similarity-based style reuse, which aims to capture similarities between different webpages, brings new challenges. In order to facilitate style sharing between different webpages, we propose a tree-structured style graph architecture for organizing the styles stored for future use. A style graph is constructed for each visited website. The style graph is organized according to the directory structure of the website. Each directory node relates to a style tree, which stores the style properties from all the visited webpages in the same sub-directory, such that they can be shared efficiently. When a new page (same for a previously visited page) is visited, we always retrieve the corresponding style tree in the corresponding sub-directory node for previously stored styles and reuse those whenever possible. If the corresponding style tree is empty because there are no previous accesses to the same sub-directory, we also propose a heuristic algorithm to search in the style graph to find reuse possibilities.

We have implemented a prototype of the proposed approach based on QtWebKit [3]. Experimental results show that our approach can speed up the webpage loading process by up to 68% (27% on average) and reduce the redundant style calculations by up to 77% for the first visit to a webpage, with almost negligible overhead.

We make the following main contributions in this paper.

- We explore the similarities between webpages and the potential of style reuse between different webpages. We find that most webpages in the same website have lots in common, not only in their appearances, but also in the style properties used. This offers us the opportunities to further optimize web browsers.
- We propose a new similarity-based style reuse technique in order to reduce the redundant calculations between webpages during web browsing. This technique can be applied to both webpage revisits and new visits.
- We demonstrate the applicability and performance of similarity-based style reuse through extensive experiments with webpages from many popular websites. We show that we are able to exploit the similarities between webpages to improve web browsing significantly. We also measure the memory overhead of our approach and the result shows that the overhead is almost negligible.
- We perform a user study and analyze the traces generated by 10 real users. We observe that the web browsing histories exhibit strong spatial and temporal locality, which demonstrates the potential of applying our approach to enhance user experience in a real environment.

The rest of this paper is organized as follows. Section 2 introduces the background of webpage processing, existing caching scheme for web browsers and Cascading Style Sheets. Similarities between webpages are explored in Section 3. The similarity-based style reuse technique is presented in Section 4. In Section 5, we report our implementation and experimental results. User study is performed in Section 6. We discuss limitations of our work in Section 7 and present related work in Section 8. At last, we conclude the paper in Section 9.

2. BACKGROUND

In this section, we first give a brief introduction to the procedure of opening a webpage. We then introduce the existing caching schemes for web browser and give an overview of Cascading Style Sheets (CSS), which will be used heavily in this paper.

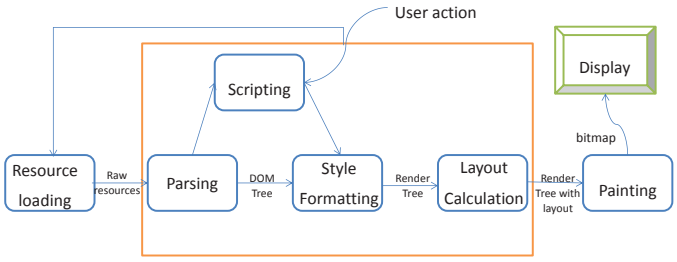

Figure 1: The process to open a webpage

2.1 The process to open a webpage

Most web browsers have similar frameworks and usually process the webpages with the same procedure. Figure 1 shows the typical work flow of a web browser. The web browser retrieves related resources from the remote web servers or local storage, including HTML documents, Cascading Style Sheets(CSS), pictures, audio files, JavaScript files, and so forth. The HTML data is the main resource, which is parsed into a DOM (Document Object Model) tree, which represents the structure of the HTML documents. Contents referenced by the URLs in HTML documents are sub-resources, which are fetched and added to the DOM tree subsequently. Style formatting operations calculate the presentations (e.g. color, font-size) for each DOM element. Layout calculations compute the position for each element in order to render them. These operations occur concurrently and the webpage is gradually drawn to the screen. After completion of loading the initial webpage, user interacts with webpages by triggering JavaScript executions. The JavaScript execution may modify the DOM tree, which would in turn cause the re-computation of the style properties and layout.

Style formatting and layout calculation account for most of the webpage processing time and they are the key issues leading to poor web browsing performance. The previous studies [9, 29] show that style formatting and layout calculation account for more than 50% of the webpage processing time. We also have profiled the performance of WebKit [6] based on OProfile [2], and the result is similar. One reason is that Cascading Style Sheets are widely and heavily used in modern webpages. Because CSS matching process is sequential and recursive, it makes the whole procedure time-consuming. Meanwhile, because the layout results are sensitive to the style properties, any changes to style properties may lead to the re-calculation of the layout, which is also a recursive process.

JavaScript execution is no longer the bottleneck for web browsers thanks to the development of JavaScript optimization techniques. For example, with the adoption of tracing JavaScript Just-In-Time (JIT) compiler in Firefox [1], the performance of JavaScript has improved significantly [7]. Previous work [16] proves that optimizing JavaScript execution on their test sites would only reduce 7% of the total CPU time at most, while parsing related tasks for JavaScript spends much more time.

2.2 Caching in Web Browsers

Caching is a well-known approach to deal with I/O bottlenecks. Frequently visited data is stored so that future requests for the same data can be served faster. The traditional caching scheme for web browsers is to store frequently visited web resources (e.g. pics, CSS files, JS files) locally, which can save the round-trip-time (RTT) for future visits, reducing network traffic as well as improving browser performance.

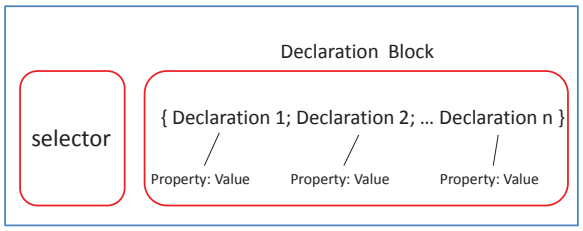

Figure 2: The general CSS rule structure

Although straightforward and effective, the traditional caching scheme faces many challenges. On one hand, the miss rate of the traditional scheme is high because more and more web contents become dynamic. Even increasing the cache size to infinite only reduces 10% of the cache misses [23]. On the other hand, the process of re-validation greatly reduces the effectiveness of the browser cache [24]. The cached resources have two states, either *fresh* or *expired*. The expiration time of resources is indicated in the HTML header. If the state is fresh, the resource can be reused without confirming with the server. If the state is expired, the browser needs to connect to the server to confirm whether the resource can be used or not. In this process, the expired resources bring in at least one RTT. Re-validations account for a large portion of cache requests from browser, thus greatly reducing the effectiveness of the browser cache [23].

Zhang *et.al* proposed Smart Caching [29] to cache the intermediate results in style formatting and layout calculation stages. The cached results can be applied in subsequent processing of the same data to avoid repeated local computations. Though dynamic webpages change frequently, the cached results may be still useful for the unchanged parts. Thus this caching scheme is effective for webpage revisits.

However, the majority of the webpage visits are new visits. For example, 75% of the webpages visited are new visits for mobile user in a recent study [24]. Based on our experimental results, different webpages in the same website share a lot of similarities. If we cache the intermediate results in computation-intensive stages of webpage processing, the results may be useful for the identical components across similar webpages.

In this paper, we propose a similarity-based caching and reusing scheme to cache and share intermediate results across similar webpages. We only cache the style results, excluding the layout results. Because layout calculation is typically content-related, the layout results are sensitive to the actual contents and style properties. Any modification to the DOM tree will lead to invalid layout cache. Different webpages in the same website may share a large portion of style properties, however, the contents are typically different. Meanwhile, CSS matching process accounts for the largest portion of the webpage processing time [7]. Thus caching and reusing the style properties across webpages have the most potential to improve the web browsing performance.

2.3 CSS Overview

CSS is used to separate the presentation from the contents for webpages. Every webpage corresponds to a style sheet, which contains a series of CSS style rules. The style rules specify the fonts, colors or other presentations of the corresponding DOM element. Each CSS rule contains two parts: the selector and the declaration block.

Figure 2 presents the general structure of CSS rules. The selector determines which kind of elements match this CSS rule.

Same resources	Different resources
load(1).css	50px-Mergefrom.svg.png
Load.css	170px-Samsung_Galaxy_S_III_Pebble_Blue_Wikipedia.png
load(1).php	Cloud_computing_layers.png
Load(2).php	400px-Cloud_computing.svg.png
Load.php	395px-Cloud_computing_types.svg.png
Index.php	325px-CloudComputingSampleArchitecture.svg.png
geoiplookup	30px-Commons-logo.svg.png
magnify-clip.png	20px-Wiki_letter_w_cropped.svg.png
poweredby_mediawiki_88x31.png	16px-Folder_Hexagonal_Icon.svg.png
search-ltr.png	12px-Commons-logo.svg.png
wikimedia-button.png	

Figure 3: The similarities of two Wikipedia webpages

There are many kinds of selectors, such as tag selectors, ID selectors, Class selectors, or complicated descendant selectors and sibling selectors, etc. The declaration block contains one or more declarations. Each declaration consists of a property and a value.

For example, a simple CSS rule `h1 { color:blue; font-h1 size:10px; }` contains a selector "h1" and a declaration block `{ color:blue; font-size:10px; }`, which contains two declarations. The meaning of this CSS rule is: set the color of the text within "h1" as blue, and set the font size as 10 pixel.

To calculate the style properties of a DOM element, a CSS matching process is needed. Each CSS rule is checked to determine whether the rule is matched to the element or not. This process is executed sequentially and recursively in order to get a proper result. Thus this process is computation-intensive and time-consuming. The CSS rule-matching component accounts for the largest portion of the webpage processing time [7].

CSS allows the style characteristics to be shared across multiple webpages. On one hand, it could reduce the repetitions in specifying the styles for each webpage. On the other hand, it is easy to keep the style consistent for all webpages in the same website. In current web design, CSS is often shared between webpages in the same website, which offers us the opportunity to exploit this feature to optimize web browsing performance.

3. SIMILARITIES BETWEEN WEBPAGES

In this section, we explain and evaluate the similarities between different webpages, which is the key motivation behind our proposed approach.

Many webpage visits are from the same website, most of them from the same sub-domain or same sub-directory. These webpages may share many common resources (eg. CSS files, JS files, pictures), possess similar webpage structure and similar (or even the same) styles. For example, in Figure 3, the two different Wikipedia webpages share a lot of similarities. From the visual point, they have the similar appearance and same strucutre, including the same navigation bar, and the same header and footer components. After analyzing their web resources, we find out that all the CSS files and JavaScript files they used are the same. Besides actual contents, only the relevant pictures are different.

Table 1: Similarities between webpages

website	webpages	HTML repetition ratio	style matching ratio
Amazon	item pages	33.7%	74.8%
Baidu	result pages	51.0%	79.5%
Bing	result pages	71.0%	54.6%
Youtube	video pages	27.1%	58.5%
QQ	news pages	53.1%	68.9%
Youku	video pages	32.9%	84.3%
Taobao	item pages	45.3%	63.3%

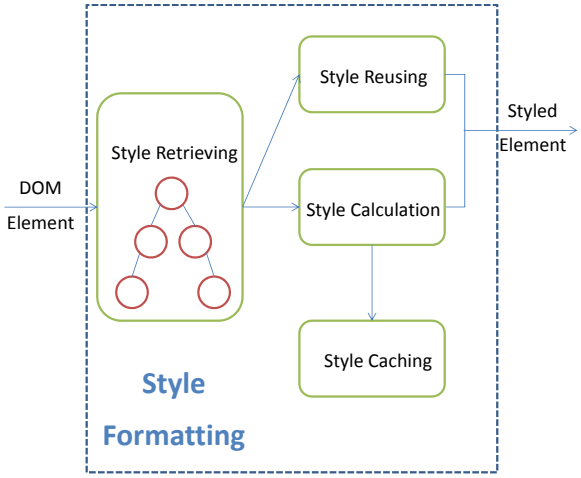

Figure 4: The procedure of similarity-based style reuse

The reason of so many similarities exist in current webpages is because that many webpages are generated based on web templates [5]. In modern web design, templates are widely used to keep the uniformity of the webpages and separate the presentation from the contents. However, the template brings in many redundant data along with the convenience. As shown in previous research [5], up to 40% of all data on the web are actually templates, which causes not only redundancies in Internet traffic, but also redundancies in webpage processing.

In order to further explore the similarities between webpages, we analyze both the HTML data repetition ratio and style matching ratio of elements for a set of webpages in the same sub-directory. We use webpages from eight of the top websites from Alex [4] in our experiments. The results are shown in Table 1. The HTML data matching ratio refers to the percentage of the same HTML data (including the JS code inside HTML document) shared between different webpages. The style matching ratio refers to the percentage of style properties shared between different webpages. The HTML data repetition ratio varies, because the contents of different webpages change a lot. However, the webpages from the same website share more than half of their style properties, up to more than 80%. This result indicates that although their contents varies, the styles keep unchanged for many webpages.

4. SIMILARITY-BASED STYLE REUSE

In this section, we describe the proposed *similarity-based style reuse* technique in detail, including style storage and management techniques, style retrieving and matching algorithms.

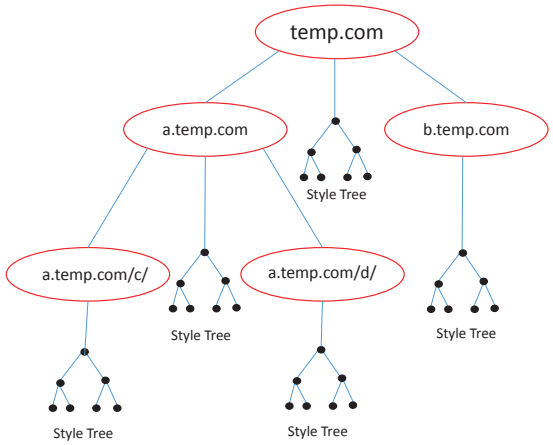

Figure 5: An example of style graph

Figure 4 presents an overview of the proposed similarity-based style reuse approach. We cache all the style calculation results during style formatting in a tree-structured architecture, which will be explained in detail next. When the style of a DOM element needs to be calculated, we first retrieve the style properties from the cached style trees. If matched style nodes are found in the style tree, the style properties can be reused without calculation. If no corresponding style nodes can be found in the style tree, we calculate the style properties for this element and cache the results in order to benefit future visits.

4.1 Style Storage & Management

In order to facilitate style sharing between different webpages, we propose a tree-structured *style graph* architecture to organize the style storage. A style graph is constructed for each visited website.

The style graph is organized according to the directory structure of the corresponding website. An example style graph is shown in Figure 5. There are two kinds of nodes in a style graph: *directory node* and *style node*. Directory nodes include domain nodes, sub-domain nodes and sub-directory nodes. They connect the entire style graph into a tree structure. They do not store any style information. Each directory node relates to a *style tree*, which stores the styles of webpages visited in the same sub-directory, such that they can be shared efficiently. All the nodes in a style tree are style nodes, which store the style properties of the corresponding DOM elements.

The architecture of a style tree is similar to a DOM tree. However, it only keeps the structure information of the DOM tree and records the style properties for the corresponding DOM elements. All webpages in the same sub-directory share a common style tree. The same sibling style elements are merged into one element, thus reducing the storage space and the complexity of style retrieving.

However, some extreme situations might happen. For example, two webpages in the same directory share a common style tree, but they are totally different, even the root elements cannot be matched. In this case, the root element cannot be merged into the style tree, let alone other elements. In order to solve this problem, a *meta-root node* is created in each style tree. The meta-root node does not store any style information. Its only purpose is to connect all the different root elements.

For each style node, we use a triple `<TagName, ID, Class>` as the *node identifier*. For each DOM element, the style properties

can be reused only if the node identifier and the path to root are matched with the style node.

Compared with the style storage method in Smart Caching [29], our proposed style graph structure is more efficient. In Smart Caching, each webpage has a corresponding style tree. However, many webpages are only visited once, for example, people seldomly access a news webpage for a second time. Thus the cached style tree would never be used again and it is a waste of resources. In contrast, all the webpages in the same sub-directory share a common style tree in our approach. The same elements are compacted to save the storage space. According to our statistics, the style graph structure can save more than 60% of the storage space on average compared with Smart Caching. When more webpages from the same sub-directoy are visited, the average style storage space for each webpage is decreased. Meanwhile, the style tree keeps the structure information of the DOM tree, which is efficient for style retrieving and matching.

4.2 Style Retrieving

Each style tree stores the styles from all the visited webpages in the same subdirectory, such that they can be shared efficiently. When a page is visited, we always check the corresponding style tree for previously stored styles and reuse those whenever possible. If the corresponding style tree is empty because there are no previous accesses to the same subdirectory, we also propose an algorithm to search in the whole style graph to find reuse possibilities. The process of style retrieving can be divided in two steps: style graph searching and style matching.

4.2.1 Style Graph Searching

When a page is visited, we always retrieve the corresponding style tree to reuse previously stored styles whenever possible. When the corresponding style tree is empty, we propose a heuristic method to find reuse possibilities in the whole style graph.

We introduce the concept of *webpage distance*. A website is typically organized as a directory structure, which can be represented as a tree structure. Any webpage can be attributed to a specific directory. The distance of two webpages is the distance of their corresponding directories, which refers to the number edges from the starting directory to the ending directory.

We assume that the shorter the distance of two webpages, the higher similarities they have. The intuition behind this is that, the webpages in the same sub-directory have higher possibility to share more common characteristics than webpages in different directories. Besides, the webpages in the same sub-domain have higher possibility to share more similarities than webpages in different sub-domains. While this assumption is not necessarily correct for every website, it offers a heuristic way to find reuse possibilities.

Based on this assumption, we use a *breadth-first search (BFS)* algorithm to search for non-empty style tree when necessary. When visiting a webpage, if the corresponding style tree is empty, we use the BFS algorithm to choose the nearest non-empty style tree. The algorithm is very efficient because the number of style trees is very limited. While the resulting style tree is reused, we cache the reused styles in the corresponding style tree for the new webpage, such that for future accesses to the webpages in the same directory, searching is not needed because the style tree is not empty any more.

4.2.2 Style Matching

Style matching is the method to compare the DOM element with the cached style nodes. If the nodes can be matched, the cached style properties can be reused directly without calculation.

When a webpage is loaded, the DOM tree is constructed in a pre-order manner, thus the parent nodes are always processed before the children nodes. The style properties of a DOM element are not only related to the CSS rules, but also related to the parent-children hierarchy. If the parent node cannot be matched, the style properties of all the children cannot be reused from the cache and should be re-calculated.

The style tree can be very large because many visited webpages may contribute to the same tree. Even so, the process to retrieve the style tree can be very efficient. When matching a DOM element, we only need to find the style node "*sp*" corresponding to its parent node, and compare the DOM element with the children elements of "*sp*". The style matching algorithm is described as below.

```
STYLECAL(DOMELEMENT D, STYLETREE T)
 1  if d = ROOT NODE
 2    then
 3         t ← COMPARE(d, T.root)
 4         if t = TRUE
 5           then d.style ←(T.root).style
 6         if t = FALSE
 7           then calculate the style for d
 8                StyleCache(d)
 9    else
10         find d's parent DOM node Pd
11         find Pd's style node s
12         for sc in children of s:
13             do
14                t ← COMPARE(d, sc)
15                if t = TRUE
16                  then d.style ←sc.style
17                       exit
18         if t = FALSE
19           then
20                calculate the style for d
21                StyleCache(d)
```

4.3 Handling CSS Inconsistencies

Our approach is based on the assumption that when two nodes have the same identifier and parent-child relationship, we could reuse the cached style properties. The assumption is reasonable in practice since it is uncommon to create two different styles with the same node identifier, or modify an existing style without changing its node identifier.

However, considering the rare situation when two pages have matching DOM elements but different CSS rules for these elements. The proposed approach will render the second page incorrectly, because it reuses the CSS rules from the first page.

To address this situation, we need to be able to detect the inconsistencies in the CSS rules for the styles with the same style node identifier. When we reuse a cached style, we record the corresponding matched CSS rules in the style tree. When a new page is being loaded, we can identify which rules are missing and which rules are newly added compared with the cached rules. If there are missing rules or newly added rules, it means that an inconsistency has been detected. We then eliminate the missing rules and add the newly matched rules to the matched rules list for each node, and recalculate the style properties. The new style calculation results are cached and they will be reused if the same node identifier and parent-children hierarchy are matched in the future.

Table 2: Websites and webpages used in experiments.

category	website	webpages
news	www.bbc.com	10 news pages
news	www.qq.com	10 news pages
search engine	www.google.com	10 result pages
search engine	www.baidu.com	10 result pages
e-commerce	www.amazon.com	10 item pages
e-commerce	www.taobao.com	10 item pages
social network	www.facebook.com	10 personal profile pages
social network	www.twitter.com	10 personal profile pages
online video	www.youtube.com	10 video pages
online video	www.youku.com	10 video pages

5. EXPERIMENTAL RESULTS

We have implemented a prototype of the proposed approach based on QtWebKit [3] (version 2.2.0) in Windows, which is a Qt-based WebKit layout engine. Our experimental machine is a mainstream PC with an Intel Dual Core 3.30GHz processor and 6GB of DDR2 RAM running Windows 7 Professional with the latest patched installed. Because the window size of the web browser can affect the performance of loading a webpage, we fix the window size at 800 by 600 pixels. We also write a script to test the webpages automatically. Because QtWebkit used in our experiments is a basic and open-source web browser, its speed cannot be compared with the heavily optimized production web browsers in the market, such as Google Chrome, Microsoft Internet Explorer, or Mozilla Firefox. However, the purpose of our experiments is showing the relative performance improvement, which should remain in a similar range when applied to other browsers.

5.1 Overall Performance Improvement

In our experiments, we select ten popular websites in five categories from Alexa [4]. For each website, we randomly choose 10 content pages in the same sub-directory as the testing webpages. The list of websites and webpages used are shown in Table 2.

Because networking speed is typically out of the control of web browsers and network conditions vary significantly during our experiments, we fetch all the webpages and related resources locally and turn off the Internet connection in our experiments. There may still exist some JavaScript asynchronous requests that cannot be fulfilled during the experiments, which could affect the absolute numbers in the results, but it would not affect the relative comparison between different sets of results.

We measure the time period for the whole page loading phase. In WebKit, both the start and finish of page loading send signals, thus the page loading time is calculated as the interval between the *load-start* and *load-finished* signals.

For each website, we test the performance of opening any two pages in different sequences with and without applying the proposed similarity-based style reuse method. Each case is tested 10 times and the average data is used in the presented results. We present the loading time improvements in Figure 6. We also show the matching ratios for each website in Table 3.

In Figure 6, the results indicate that the performance of the proposed approach varies for different websites. The geometric mean of speedup for loading time is 27% for all webpages. For webpages such as BBC and Amazon the speedup can reach as high as more than 60%. For webpages of search engines like Google and Baidu, the improvement is not obvious and we only can see a speedup about 10% compared to the original browser. In Table 3,

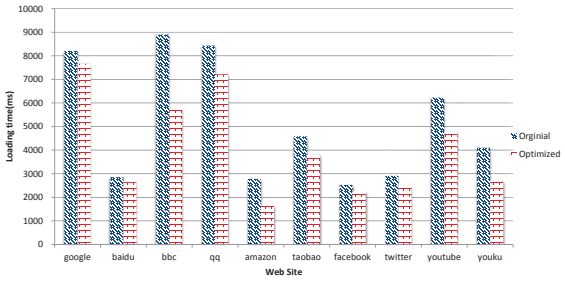

Figure 6: Loading time improvements for different websites.

Table 3: Style Matching ratios for different websites.

website	total operations	matched operations	matching ratio
www.google.com	3009	1357	44.6%
www.baidu.com	464	360	77.7%
www.bbc.com	1023	392	38.8%
www.qq.com	1259	583	39.9%
www.amazon.com	2079	1105	53.4%
www.taobao.com	1978	1397	70.7%
www.facebook.com	1120	256	22.1%
www.twitter.com	855	93	10.1%
www.youtube.com	970	497	51.2%
www.youku.com	1901	1200	41.7%

we can see that the style matching ratios range from 10% to nearly 80% for webpages across different websites. The geometric mean of the style matching ratios is 41.7%. The matching ratios for fresh new pages of Facebook and Twitter are relatively low.

Because webpages from different types of websites typically show relatively different behaviors, we also compare the data from different categories, which is shown in Table 4.

From the comparison, we can see that social network websites have the lowest matching ratios, because the personal profile pages of social network websites differ significantly. On one hand, every user could customize his own profile page with different templates provided by the website, which causes the styles different between webpages. On the other hand, each fresh news entry in the webpages of Twitter usually has a unique "ID". Although most of them have similar styles, we still can not reuse them because their IDs are different.

The results show that search engine websites have a high matching ratio but the performance improvement is relatively low. We analyzed these webpages and identified two reasons. First, there are a lot of Javascript executions in the webpages, which occupy a large portion of the loading time. Second, these webpages usually have fewer elements than webpages in the other website categories. Thus the style formatting process only occupies a small portion of the whole page loading time.

Table 4: Performance gains for different website categories

Website Category	Speedup	Matching Ratio
search engine	9.1%	58.9%
news	36.8%	39.4%
e-commerce	42.2%	61.4%
social network	13.9%	16.3%
online video	41.1%	56.9%

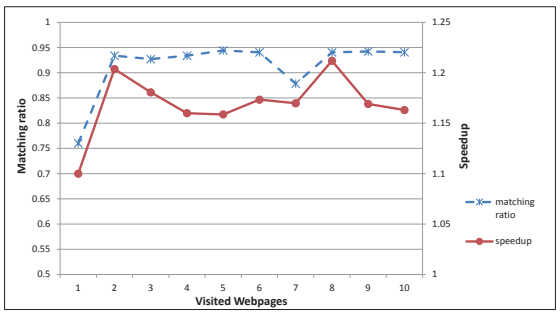

Figure 7: The speedups and matching ratios for 10 consecutively visited webpages

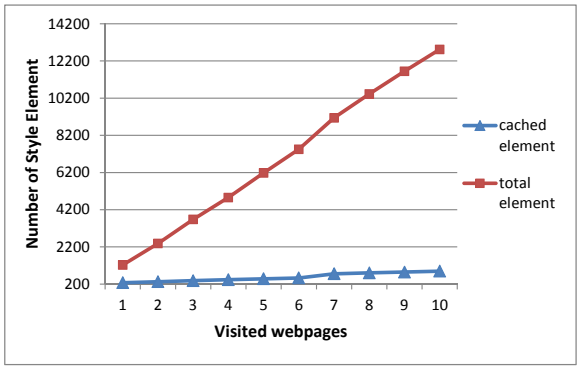

Figure 8: The number of total and cached style elements for 10 consecutively visited webpages

The webpages of news, e-commerce and online video websites usually have many elements with the same style properties, thus the performance can be improved significantly. The matching ratio is typically over 50%, and the speedup is more than 36%.

The results show that:

- For webpages using heavy CSS styles, our approach can improve the performance greatly. Our approach is suitable for webpages whose style formatting process accounts for a large portion of page loading time.
- The performance of our approach is also related to the design pattern of webpages. Too many unique IDs and classes in the webpages will lead to low matching ratios and low performance gains.

5.2 Consecutive Visits to Similar Webpages

In this experiment, we test the performance of opening a series of webpages in the same sub-directory. The purpose is to test whether we can eliminate more redundant calculations when more similar webpages are cached. At the same time, we also calculate the size of the style trees as an indicator of the incurred overhead.

We choose 10 news webpages in the sub domain "news.sina.com.cn". The webpages are visited in a particular order consecutively. There is no style cache available when the first webpage is accessed.

The speedups and matching ratios of these 10 continuous visited webpages are shown in Figure 7. The matching ratio for the first webpage is lower than the others because there is no style cache when the first webpage is visited, and the matched elements are matches within the same page. Because there is a similar

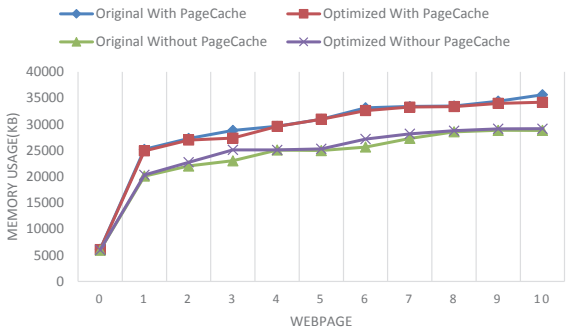

Figure 9: Memory usage comparison between the original and optimized browser with/without PageCache

mechanism in WebKit to reduce the redundant computations within a webpage, although the internal matching ratio is 75%, the speedup is low. There are no obvious performance gains with more webpages visited for the following webpages. The speedup keeps in a relatively stable level between 15% to 20%. The matching ratios remain at about 90%.

Then, we evaluate the number of cached style elements and the total number of elements while visiting more webpages. The result is shown in Figure 8, which is an accumulated graph. The style tree is initially empty and it grows in a relatively slow speed as different elements are added. Compared with the total elements, the number of cached elements in the style tree is much smaller. It means that the styles of many elements are the same and we have actually eliminated those redundant style calculations with similarity-based style reuse.

5.3 Memory Overhead

Currently, our approach stores the cached style results in the memory of the browser process. Thus we compare the memory usage of QtWebkit with and without the proposed caching scheme to understand the overhead of our approach. Similar to the previous experiments, we choose 10 webpages in the sub-domain "news.sina.com.cn" and open them consecutively. We record the memory usage after each visit respectively. Because QtWebkit is compiled in the Windows environment, we can use the Windows Task Manager to monitor the memory usage of QtWebkit.

The result is shown in Figure 9. We can see that memory usage of the optimized browser is almost unchanged compared to the original browser. By closer inspection, we can even see the total memory usage is reduced a little bit with the optimization enabled. According to common sense, the memory usage of optimized browser should be higher than the original browser because we cached extra data structures such as style graphs in the memory. The result is somewhat surprising and we further investigated the reasons.

While caching the style nodes in our approach, we have not created new style nodes but pointed them to the previously created style nodes using a pointer. These style nodes are referenced by pointers thus they would not be destroyed even the old pages are closed. Furthermore, there is a cache mechanism called "Page Cache" in WebKit. It is used only when a user clicks the back or forward button. When a user navigates to a new page, the previous page is not thrown out completely, instead it is placed in the Page Cache. The Page Cache mechanism makes clicking the back button almost instantaneous. To use Page Cache, the results of previously visited webpages are stored in the memory, including the style

Table 5: Browsing history traces of different users.

USER	History Items	USER	History Items
User1	535	User2	25331
User3	1534	User4	1904
User5	8539	User6	33056
User7	562	User8	5097
User9	8428	User10	25369

nodes. Consequently, our approach have no memory overhead compared to the original browser with Page Cache. Due to the consecutive visits to similar webpages, some style nodes can be reused, thus the memory usage of the optimized browser can be even lower than the original browser.

In another set of experiments, we turn off the Page Cache mechanism in WebKit by modifying the source code for both the original browser and optimized browser. We re-run the experiments and the result is also shown in Figure 9. It shows that the memory usages of both the original browser and optimized browser are decreased about 5,000 KB after turning off the Page Cache. The memory usage of the optimized browser is slightly higher than the memory usage of the original browser, with an increase of 660 KB on average. This is expected because we cached the style nodes in the optimized version, which are not preserved in the original browser with Page Cache turned off.

Nonetheless, the growth of memory usage is negligible compared to the total memory usage of the browser in all cases.

6. USER STUDY

In this section, we perform a user study and analyze web browsing traces generated by real users. As stated above, our approach is demonstrated effective for webpage revisits and new visits of similar webpages in experiments. The intention of this user study is to understand the browsing behaviors of real users and verify the validity of our approach.

We choose ten volunteers in our lab and analyze their web browsing histories collected from different browsers including Internet Explore, Chrome and Firefox. The time period varies from two days to more than three months, and the number of visited URLs varies from 535 to 33056, as shown in Table 5. All the browsing traces are continuous.

In order to understand the revisit ratio in their web browsing histories, we analyze the percentage of visited URLs which occurred within a specified period of time after the last visit to the same URL/domain/directory. Figure 10 shows the analysis results for different time periods, which are set as 2 minutes, 10 minutes, 1 hour and 1 day, respectively.

We see that the revisit ratio to the same domain is more than 90% within one day. Even after shrinking the time period to two minutes, the revisit ratio to the same domain is still nearly 70%. Furthermore, the revisit ratio to the same directory is more than 50% within two minutes on average. However, the revisit ratio for the same URL is much smaller, which is only about 10% within two minutes.

We make the following observations from the results:

- The web browsing history exhibits very strong spatial locality. The users are likely to visit URLs which are highly related (i.e., closer) to the current URL in the near future, such as URLs in the same domain or same directory.
- The web browsing history exhibits very strong temporal locality. We see that there is almost a 50% chance for a user to access a webpage in the same domain/directory within two minutes.
- The revisit ratio to the same URL is much lower than the ratio of visiting to the same directory or same domain. Thus the traditional cache scheme for the same webpage works only for a small portion of webpage visits.

These observations show that webpage revisits in the same domain/directory within a short period occur very frequenlty, which means great potential to enhance user browsing experiences using our approach, which is able to take advantage of similarities between recently visited webpages.

7. DISCUSSIONS

The experimental results with our prototype implementation demonstrate the effectiveness and applicability of similarity-based style reusing. In this section, we examine possible limitations in our current design and discuss potential future improvements.

Cache Storage.

Our approach currently stores the cached style results in the memory of the browser process, rather than serializing the results to files on the disks. The reason is that many repetition visits happen within a very short time period (as short as 2 minutes) after the last visit to the same URL/domain/dir, as demonstrated in Section 6. Though we cache the style results in the memory, our approach has almost no memory overhead comparing with the original browser, which is demonstrated in Section 5.3. The caches could easily be serialized and stored in permanent files on the disks if needed.

Network Influence.

Networking speed is typically out of the control of web browsers. When networking speed is fast, the process of local processing accounts for a majority of the web browsing time, thus our approach is able to improve the performance of web browser significantly. However, in the case of poor networking speed, the process of resource loading only accounts for a small portion of the total page loading time, thus our approach may not improve the overall browsing performance obviously. In the evaluation Section 5, we have not taken the networking speed into consideration because networking speed varies greatly at each test, making it is impossible to accurately measure the impact of our approach.

Mobile Web Browser.

As mobile devices become more and more popular, many people are using mobile browsers to access the Internet. The redundant calculations might cause a greater problem on mobile browsers. Although we have only implemented the proposed technique on a PC environment, it can be easily incorporated to a mobile browser. As the processing speed of mobile devices is typically slower, we expect that the proposed technique can achieve a better improvement for mobile browsers and has the potential to reduce the power/energy consumption as well.

Redundant Data Transmission.

Webpage templates are widely used in current web designs. Webpage templates refer to pieces of HTML code common to a set of webpages usually adopted by content providers to enhance the uniformity of Websites. The templates cause the contents to be replicated in multiple webpages and occur in large volumes. According to previous studies [20, 21], the templates represent over 40% of data currently available on the Web. Fetching and

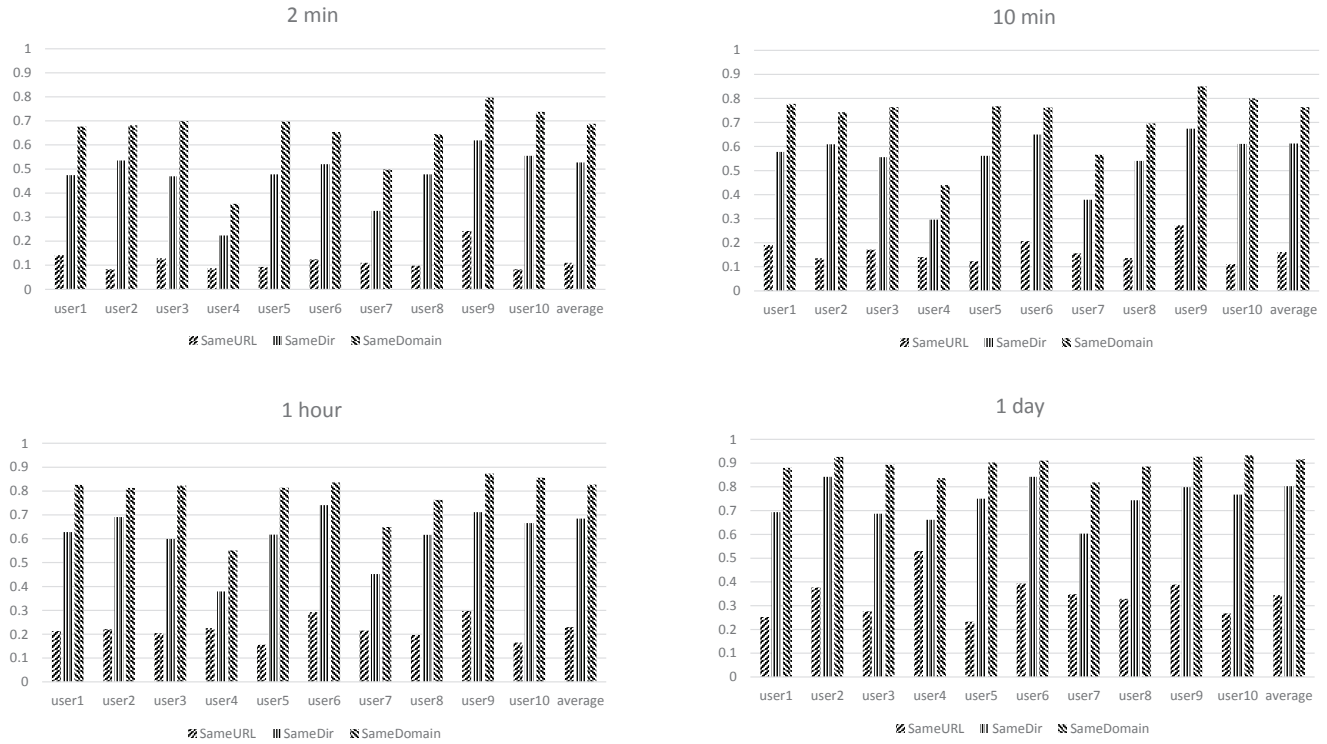

Figure 10: Revisit ratios to the same URL/domain/directory for 10 real users within different time period (2min/10min/1hour/1day).

processing the templates is likely to lead to a waste of bandwidth and computation resources. It offers the opportunity of reducing both the redundant computations and network traffic.

8. RELATED WORK

Web browser is one of the most important applications on devices connected to the Internet, thus improving the performance of web browsers is very important. Researchers and industry organizations propose various methods to build a better web browser. The authors in [22] have surveyed optimizing approaches for web browsers. The approaches include caching optimization [29, 18, 19], parallelization [7, 17, 12, 9], prefetching [15, 8], speculative loading [24] and utilizing the power of cloud computing to enhance web browsers [25, 27, 28, 26, 13, 14, 10, 11].

Effective Caching.

The typical caching scheme in current web browsers is to cache the web resources, including HTML data, pictures, CSS files and JS files. Smart Caching [29] presents a novel caching method to cache the intermediate results in style formatting and layout calculation stages. The cached results can be applied in subsequent processing of the same data to avoid the repeated local computations. This method only addresses the redundant calculations for revisits to the same webpage.

Parallelization.

Parallelizing the computation-intensive steps in webpage processing can improve the performance of browsers. Jones et al. [9] analyze the browser and discuss how parallelism can make the browser more responsive and energy-efficient. They describe the design of a parallel web browser, and analyze the possibility and algorithms to parallelize each browser component, including parallelizing the fronted, parallelizing page layout and parallelizing the scripting.

Prefetching.

Webpage prefetching attempts to predict which webpages user will visit in the near future and download the predicted webpages before actual visits. If the prediction is correct, the browser can use the resources preloaded locally. For example, PocketWeb [15] tries to predict user accesses by analyzing their access history. The web visiting history of a user is used to train the user access model using a machine-learning approach. The result shows that for 80% to 90% of the users, the model can accurately prefetch 60% of the URLs within 2 minutes before requests.

Speculative Loading.

The motivation of speculative loading [24] is similar to our work. They find that webpages in the same website usually share a lot of common web resources. Thus they propose a speculative loading method to predict which resources may be useful when visiting a webpage and the predicted resources are loaded along with the main resources, thus saving the round-trip-time. The difference is that they utilize the similarities between webpages to speedup resource loading, while we use the similarities to improvement the performance of webpage processing.

Cloud-Based Optimizations.

The performance of web browsers can be improved with the aid of cloud computing. The cloud can act as an agent of the web browser, and all traffic between the web server and the mobile browser passes through the clouds. The cloud can preprocess and compress the web contents before sending the results to web

browsers. Some current browsers use this approach, such as Deepfish [26], Opera mini [27] and Amazon Silk [25]. Besides, web browsers can also offload heavy computations in webpage processing to a cloud. Many researches have studied computation offloading [11, 10], which can also be applied to web browsers.

9. CONCLUSIONS

As webpages become more and more complex, the performance of browsers has become critical for users to have a satisfactory browsing experience. In this paper, we explore the potential to improve webpage processing performance by exploiting the similarities between webpages.

We propose a new similarity-based style reuse technique in order to reduce the redundant calculations between webpages. A style graph structure is proposed to cache the style calculation results efficiently. We also propose a heuristic algorithm to search for style reuse possibilities in the style graph.

Experiments on webpages from popular websites show that the proposed method can achieve a speedup of up to 68% (27% on average) on the whole page loading time and reduce the redundant style calculations by up to 77% for the first visit to a webpage. Besides, we also perform a user study and analyze traces generated by real users, which shows that our approach could work effectively in a real environment.

Acknowledgment

This work is supported in part by the National Basic Research Program of China (973) under Grant No. 2011CB302604, the National High-Tech R&D Program (863) under Grant No.2011AA01A202, the National Natural Science Foundation of China under Grant No.61103026, 61121063, 91118004 and the Key Program of Ministry of Education, China under Grant No.313004.

10. REFERENCES

[1] Firefox. http://en.wikipedia.org/wiki/Firefox.

[2] Oprofile. http://en.wikipedia.org/wiki/OProfile.

[3] Qtwebkit. http://trac.webkit.org/wiki/QtWebKit#.

[4] The top 500 sites on the web. http://www.alexa.com/topsites.

[5] Web template. http://en.wikipedia.org/wiki/Web_template.

[6] The webkit open source project. http://www.webkit.org/.

[7] C. Badea, M. R. Haghighat, A. Nicolau, and A. V. Veidenbaum. Towards parallelizing the layout engine of firefox. In *Proceedings of the 2nd USENIX conference on Hot topics in parallelism*, HotPar'10, 2010.

[8] C. Bouras and A. Konidaris. Predictive prefetching on the web and its potential impact in the wide area. *World Wide Web*, 7:143–179, 2004.

[9] R. B. Christopher Grant Jones, Rose Liu, Leo Meyerovich, Krste Asanovic. Parallelizing the web browser. In *First USENIX Workshop on Hot Topics in Parallelism (HotPar '09)*, 2009.

[10] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems (EuroSys '11)*, pages 301–314, 2011.

[11] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys '10)*, pages 49–62, 2010.

[12] T. Hottelier, J. Ide, R. Bodik, and D. Kimelman. Parallel web scripting with reactive constraints. In *Technical Report No. UCB/EECS-2010-16*, 2009.

[13] J. Kim, R. A. Baratto, and J. Nieh. pthinc: a thin-client architecture for mobile wireless web. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 143–152, 2006.

[14] A. M. Lai, J. Nieh, B. Bohra, V. Nandikonda, A. P. Surana, and S. Varshneya. Improving web browsing performance on wireless pdas using thin-client computing. In *Proceedings of the 13th international conference on World Wide Web (WWW '04)*, pages 143–154, 2004.

[15] D. Lymberopoulos, O. Riva, K. Strauss, A. Mittal, and A. Ntoulas. Pocketweb: instant web browsing for mobile devices. *SIGARCH Comput. Archit. News*, 40(1):1–12, Mar. 2012.

[16] L. A. Meyerovich and R. Bodik. Fast and parallel webpage layout. In *Proceedings of the 19th international conference on World wide web (WWW '10)*, pages 711–720, 2010.

[17] L. A. Meyerovich and R. Bodik. Fast and parallel webpage layout. In *Proceedings of the 19th international conference on World wide web (WWW '10)*, pages 711–720, 2010.

[18] K. Muralidhar and N. Geethanjali. Fuzzy Replacement Algorithm for Browser Web Caching. *International Journal of Engineering Research and Applications (IJERA)*, 2(3):3017–3023, 2012.

[19] S. Podlipnig and L. Böszörmenyi. A survey of web cache replacement strategies. *ACM Comput. Surv.*, 35(4):374–398, Dec. 2003.

[20] K. Vieira, A. L. Costa Carvalho, K. Berlt, E. S. Moura, A. S. Silva, and J. Freire. On finding templates on web collections. *World Wide Web*, 12(2):171–211, June 2009.

[21] K. Vieira, A. S. da Silva, N. Pinto, E. S. de Moura, J. a. M. B. Cavalcanti, and J. Freire. A fast and robust method for web page template detection and removal. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, CIKM '06, pages 258–267, New York, NY, USA, 2006. ACM.

[22] H. Wang, J. Kong, Y. Guo, and X. Chen. Mobile web browser optimizations in the cloud era: A survey. In *Proceedings of the 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, pages 527–536, 2013.

[23] Z. Wang, F. Lin, L. Zhong, and M. Chishtie. How effective is mobile browser cache? In *Proceedings of the 3rd ACM workshop on Wireless of the students, by the students, for the students (S3 '11)*, pages 17–20, 2011.

[24] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie. How far can client-only solutions go for mobile browser speed? In *Proceedings of the 21st international conference on World Wide Web (WWW '12)*, pages 31–40, 2012.

[25] Wikipedia. Amazon silk. http://en.wikipedia.org/wiki/Amazon_Silk.

[26] Wikipedia. Microsoft live labs deepfish. http://en.wikipedia.org/wiki/Microsoft_Live_Labs_Deepfish.

[27] Wikipedia. Opera mini. http://en.wikipedia.org/wiki/Opera_Mini.

[28] Wikipedia. Skyfire (web browser). http://en.wikipedia.org/wiki/Skyfire_(web_browser).

[29] K. Zhang, L. Wang, A. Pan, and B. B. Zhu. Smart caching for web browsers. In *Proceedings of the 19th international conference on World wide web (WWW '10)*, pages 491–500, 2010.