Web Performance Acceleration by Caching Rendering Results

Yuusuke Nakano
Graduate School of
Information and Science Technology
Osaka University
Osaka 565–0871, Japan
NTT Network Technology Labs
Tokyo 180–8585, Japan
nakano.yuusuke@ist.osaka-u.ac.jp

Noriaki Kamiyama
Graduate School of
Information and Science Technology
Osaka University
Osaka 565–0871, Japan
NTT Network Technology Labs
Tokyo 180–8585, Japan
kamiyama.noriaki@ist.osaka-u.ac.jp

Kohei Shiomoto
NTT Network Technology Labs
Tokyo 180–8585, Japan
shiomoto.kohei@lab.ntt.co.jp

Go Hasegawa
Cybermedia Center
Osaka University
Osaka 560–0043, Japan
hasegawa@cmc.osaka-u.ac.jp

Masayuki Murata
Graduate School of
Information and Science Technology
Osaka University
Osaka 565–0871, Japan
murata@ist.osaka-u.ac.jp

Hideo Miyahara
Graduate School of
Information and Science Technology
Osaka University
Osaka 565–0871, Japan
miyahara@ist.osaka-u.ac.jp

***Abstract*—Web performance, the time from clicking a link on a web page to finishing displaying the web page of the link, is becoming increasingly important. Low web performance of web pages tends to result in the loss of customers. In our research, we measured the time for downloading files on popular web pages by running web browsers on four hosts worldwide using PlanetLab and detected the longest portion in download time. We found the longest portion in download time to be Blocked time, which is the waiting time for the start of downloading in web browsers. In this paper, we propose a method for accelerating web performance by reducing such Blocked time with a cache of rendering results. The proposed method uses an in-network rendering function which renders web pages instead of web browsers. The in-network rendering function also stores the rendering results in its cache and reuses them for other web browsers to reduce the Blocked time. To evaluate the proposed method, we calculated the web performance of web pages whose render results are cached by analyzing the measured download time of actual web pages. We found that the proposed method accelerates web performance of long round trip time (RTT) web pages or long RTT clients if the web pages ' dynamic file percentages are within 80%.**

I. Introduction

Web performance, the time required from clicking a link on a web page to finishing displaying the web page of the link, is becoming increasingly important. Low web performance of web pages tends to result in the loss of customers and revenue. For example, a 2000-ms delay reduces per-user revenue by 4.5% [1].

There are three types of web performances, web server, network, and web browser. Web server performance is related to the processing time for each request and number of retransmissions. Retransmissions are required when the web server discards requests due to overflow of its queue. Network performance is related to round trip time (RTT), throughput, and number of retransmissions between a web browser and web server. The RTT has a higher impact on web performance than throughput [2].Retransmission is required when routers discard packets because of congestion. Web browser performance is related to the structure of a web browser and web page. A web browser downloads files, such as scripts and images, composes a web page, creates a bitmap, and displays it on a screen. This web browser ' s process is called rendering. The web browser attempts to download files in a parallel manor to accelerate web performance, but files of some web pages are not downloaded in parallel due to the structure of the pages. This difficulty of parallel downloading may affect web performance [1].

We detected the lowest performance among the three types and found that web browser performance is the lowest. This is the same result as Souders mentioned [1].To improve web browser performance, we propose a method for reduce rendering time by reusing cached rendering results. With this method, an in-network rendering function renders web pages instead of web browser and caches the rendering results. After caching the rendering results, the proposed method reuses the results to reduce rendering time. As a result, web performance is accelerated.

II. Detection of factors for low web performance

To find the most dominant factor for low web performance, we conducted the following investigation.

1) Measurement of download time of files in real web pages.
2) Detection of the lowest of the three types of performances among by considering the measured download time.

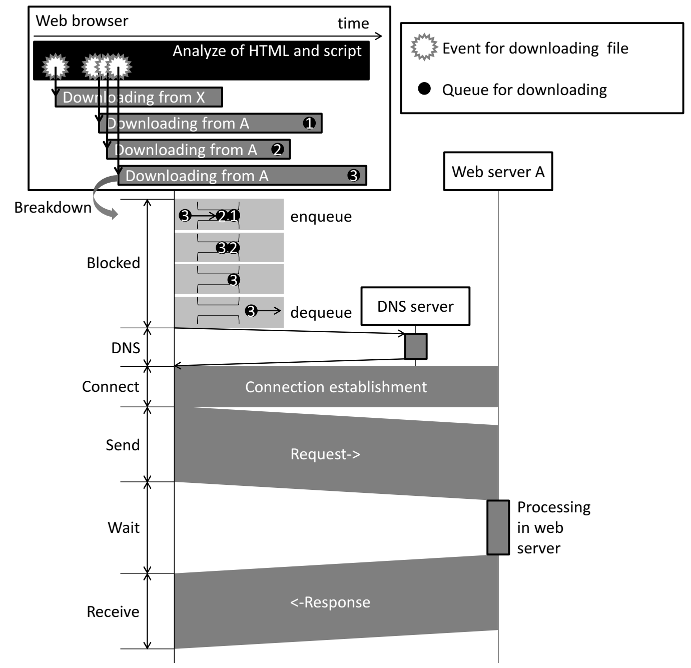

Fig. 1. Breakdown of download time

A. Measurement of download time of files on web pages

1) Measurement format: Modern web browsers, such as Firefox and Chrome, enable us to measure the download starting time of files on web pages and the time for downloading and outputting them as JSON files called HTTP Archive (HAR) [3] files. The time for downloading is divided into six parts. Figure 1 shows relationship between web browser behavior, download starting time and each portion of the download time.

2) Download starting time: A web browser analyzes HTML documents of web pages and some types of scripts defining them. At the same time, the web browser generates events to start downloading files that make up the web pages in accordance with the analysis results. As shown in Figure 1, an event for downloading a file from web server X is generated first, then three events for downloading from web server A are generated. The time for generating these events is written in the HAR file as the download starting time.

3) Each portion of divided download time: There are dependencies between files on a web page, and files are downloaded in a particular order of these dependencies. In addition, the number of web server parallel downloads is limited to a predefined number for reducing web server load, and such a maximum number of parallel downloads is set for each web browser in advance as a locked-in value. In Figure 1, the maximum number of parallel downloads is assumed to be 1 (the max number for Chrome is 6). While the web browser is downloading a file, the rest of the downloading files are enqueued and wait to be dequeued. After being dequeued, the web browser starts downloading the file. This kind of waiting time in the web browser is registered as Blocked in the HAR file. There are many factors that cause Blocked time and they are summarized by Wang et al [4]. After waiting for the downloading to start, the web browser obtains an IP address

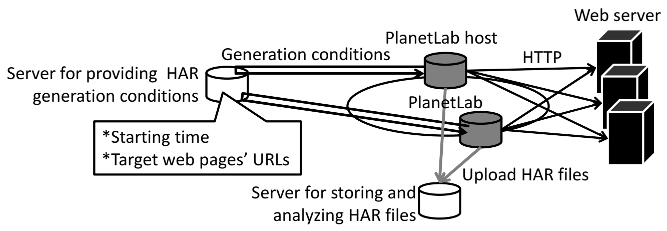

Fig. 2. HAR file generation environment

by name resolution using a domain name system (DNS) server. The time for name resolution is registered as the DNS in the HAR file. If the DNS cache in the web browsers has a result of the name resolution, the DNS in the HAR file is 0. After name resolution, the web browser establishes a connection with a web server, sends an HTTP request, waits for an HTTP response, and receives the HTTP response. The time for each process is registered as Connect, Send, Wait, and Receive, respectively, in the HAR file.

Blocked time is time spent waiting for the start of downloading in web browsers, and the DNS, Connect, Send, and Receive are time spent in the network. On the other hand, Wait is time spent in both the web server and network. The most dominant factor for low web performance is assumed to be where the rendering process spends the longest time.

4) HAR file generation environment: We generated HAR files using real web browsers for displaying web pages to detect the most dominant factor for low web performance. Figure 2 shows the HAR file generation environment. The time for downloading files that make up web pages depends on where the web browser is. To run web browsers in hosts worldwide, we used PlanetLab [5], which is an experimental environment that provides a huge amount of hosts connected to the Internet worldwide. We selected four hosts from PlanetLab and installed a web browser (Firefox) and an HAR generation program on each host. The selected hosts were in California, USA, France, Australia, and Argentina. The HAR generation program downloads a description of generation conditions, which are starting time for generation and URLs of target web pages, from a server providing the generation conditions, and operates the web browser to generate an HAR file in accordance with the generation conditions. Finally, the HAR generation program uploads the generated HAR files to a server for storing and analyzing them.

The time for downloading files of web pages also depends on which web pages the web browsers display, and it is necessary to generate HAR files from a huge variety of web pages. We made a list of 959 URLs by extracting URLs from the Alexa [6] popular web page ranking, a web site ranking providers. We extracted the top 70 URLs from each category such as news and shopping, and merged them into the 959-URL list. We generated HAR files in accordance with the list.

B. Longest time among six portions of download time

As mentioned above, we generated HAR files of target web pages by displaying them with web browsers installed on hosts in four countries and detected the longest time among the six portions of the downloading time. Table 1 lists the averages of each portion of downloading time in each host. We found Blocked and Wait times are longer than other parts of the downloading time and Blocked time is the longest and occurs when the web browser reaches the maximum number of parallel downloads.

In addition to reaching the maximum number, Blocked time also occurs when there are dependencies between files making up a web page. Wang et al. argued that there are various types of dependencies between files in a web page [4] and to start downloading a file, it is often required to finish downloading previous files. This leads to a long Blocked time.

As above, we found that the most dominant factor for low web performance is waiting time in the web browser. Reducing this kind of time can accelerate web performance.

III. Proposed method

It is assumed that the waiting time in the web browser increases when the number of files to be downloaded increases because the number of dependencies between the files increases. To reduce such waiting time in web browsers, reducing the number of files to be downloaded is effective. Our proposed method reduces the waiting time in web browsers for accelerating web performance by caching rendering results and reducing the number of files to be downloaded.

The proposed method is composed of in-network rendering functions near the network edge which render and cache the web pages and special web browsers installed in user client PCs. In-network rendering functions are implemented in the form of special server instead of web proxy and the special web browsers are implemented in the form of web browser extension. The user sends a URL of a web page he/she wants to see via the web browser to the function. Then, the function renders the web page of the URL in the same manner as ordinary web browsers. After that, the function caches the rendering results of the web page and sends the rendering results to the web browser. Finally, the web browser displays the rendering results. In this way, the waiting time in the web browser occurs as before at the first time of rendering the web page. On the other hand, when there are rendering results previously rendered for another user, all the function has to do is obtain the rendering results from its cache and send them to the web browser. As a result, the web browser can quickly display rendering results.

However, most web pages have files that dynamically change, and this means that rendering results for one user are not usable for every user. For example, web advertisements dynamically change every time they are displayed on web browsers and time lines of twitter or Facebook are dynamic. Files for such content seem to be dynamic.

TABLE I. AVERAGES OF EACH PORTION OF DOWNLOAD TIME IN EACH HOST (MS)

Part of download time \ Host	California	Argentina	France	Australia
Blocked	255.36	552.92	418.45	377.77
DNS	0.68	0.30	0.43	0.85
Connect	13.86	25.01	81.32	21.18
Send	0.02	0.01	0.01	0.02
Wait	195.10	392.28	235.04	292.07
Receive	53.10	78.61	46.43	72.43

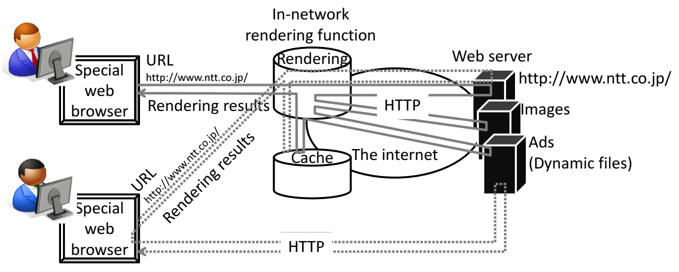

Fig. 3. Proposed method

Figure 3 shows an abstract of the proposed method which takes into account the problem of such dynamic files. As mentioned above, the in-network rendering function renders a web page in the same manner as ordinary web browsers if it renders the web page for the first time. After rendering, the function sends the rendering results to the web browser. On the other hand, if it is not the first time to render the web page, the process of the function is as follows.

- **2nd time** The function renders again and obtains the difference between the 1st and 2nd time rendering results. As a result, the function extracts a static portion of the web page and caches that portion. At the same time, the function sends the newly rendered results to the web browser. The cached rendering results contain serialized data generated in the rendering process such as DOM trees and render trees of static portions, JavaScript converted into a web browser ’ s special expression, external resources such as scripts and images, and Extensible Stylesheet Language Transformations (XSLTs) for connecting dynamic portions in HTMLs and nods in the render trees.
- **After 2nd time** The function obtains the HTML of requested URLs from the web server and sends it with the cached static rendering results to the web browser.

For more accurate extraction of static portions of HTMLs, the function can increase the number of obtaining the differences.

After obtaining the rendering results, the web browser loads them into its memory and extracts the dynamic portion from the HTML received from the function using the XSLTs. After that, the web browser renders the extracted dynamic portion by downloading files, such as scripts and images, from web servers. In this way, the proposed method reduces the number of files to be downloaded by rendering the dynamic portions and downloading the dynamic files instead of rendering and downloading entire HTML and all files. As a result, the method reduces the waiting time in web browsers and rendering time. In other words, the method accelerates web performance.

IV. Evaluation

We evaluated the proposed method before implementation.

A. Evaluation method

To evaluate the effectiveness of the proposed method, we calculated the rendering time with cached rendering results from HAR files and compared it with that without the cached rendering results. The calculation method is as follows.

1) Classification of static and dynamic files in actual web pages.
2) Calculation of rendering time for the dynamic portions of an HTML document.

We calculated the rendering time for dynamic portions of an HTML document and found how long it takes to render by reusing the cached static rendering results. We also assumed the time for obtaining the cached rendering results and that for loading them into memory are zero seconds.

We explain the details of the calculation method in the following sections.

1) Method for classification of static and dynamic files: We used file size and URLs for the classification of static and dynamic files. An HAR file has the size and URL of each file on the web page. By generating multiple HAR files for each web page and comparing the size and URLs among the HAR files generated from the same URL's web page, we determined if the files are dynamic or static.

2) Method for calculation of rendering time for dynamic portion: There are dependencies between files on a web page. The calculation method has to calculate the rendering time for the dynamic portion by taking into account these kinds of dependencies. The calculation method calculates the onload time, which is the time from the start of downloading files to finish, which are necessary for displaying the web pages. First, the method clusters the files that start to be downloaded at nearly the same time. There seems to be no dependence between files in one cluster because web browsers download the files in parallel, and all the method has to take into account is the dependencies between clusters. After clustering, the method calculates the average time for downloading files in each cluster by making the download time of dynamic files zero seconds. The method defines this average download time as each cluster ' s download time. Using the cache, the web browser downloads and renders dynamic files instead of all files on the web page and each cluster ' s download time is shortened. Because the clusters are dependent on each other, shortening of one cluster's download time affects the download start time of files in other clusters after the shortened cluster. The calculation method updates the start time of downloading files in each cluster in accordance with the changes in previous clusters' download times. In particular, the calculation method pushes the download start time of files in a cluster ahead by the sum of the changes in previous clusters ' download times. Finally, the calculation method calculates the onload time, which is the time for finishing the downloading of all files necessary for rendering (files finishing downloading until onload event generated by the web browser), and provides the updated onload time as the time for rendering the dynamic portion.

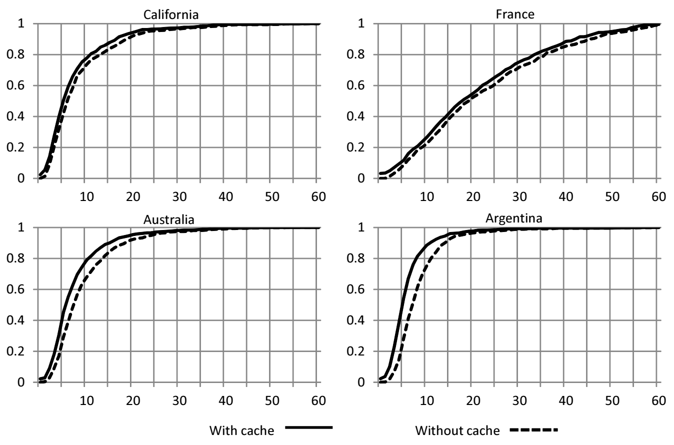

Fig. 4. Cumulative distribution of onload time by using cache of proposed method and without it (vertical axis: cumulative probability of onload time, horizontal axis: onload time (seconds))

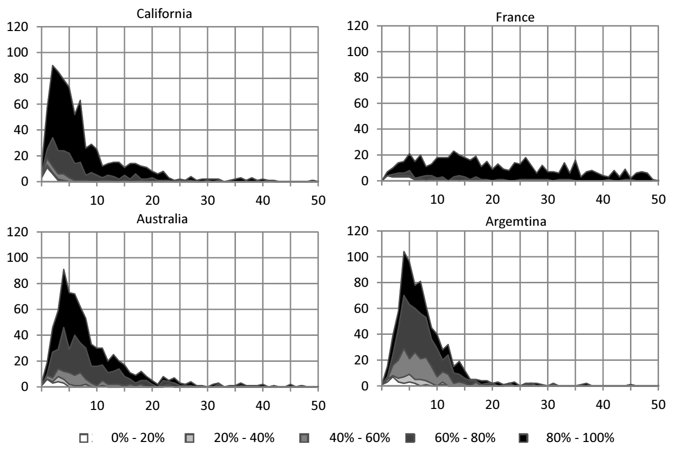

Fig. 5. Onload time histogram of every second in each host (vertical axis: number of web pages in each onload time, horizontal axis: onload time without cache (seconds), each area: percentage of onload time with cache in onload time without cache)

B. Evaluation results

We evaluated the proposed method by using the evaluation method mentioned above. We used the same HAR files as those discussed in Section 2.1.4 and classified dynamic and static files by obtaining the differences between HAR files of the same URLs generated in each host.

Figure 4 shows the cumulative distribution of onload time by using the cache of the proposed method and without using it. The onload times in Australia and Argentina were significantly shortened than those in other hosts. This is obvious because the cumulative distribution line of onload time by using the cache lies to the left of that without using it, compared with those of other hosts. It is often said that most users stop waiting for the rendering of web pages to finish after five seconds. In Argentina, the percentage of onload time shorter than five seconds increased from 28 to 45%.

Figure 5 is a histogram of onload time every second in each host. We divided the histogram into five areas and each area

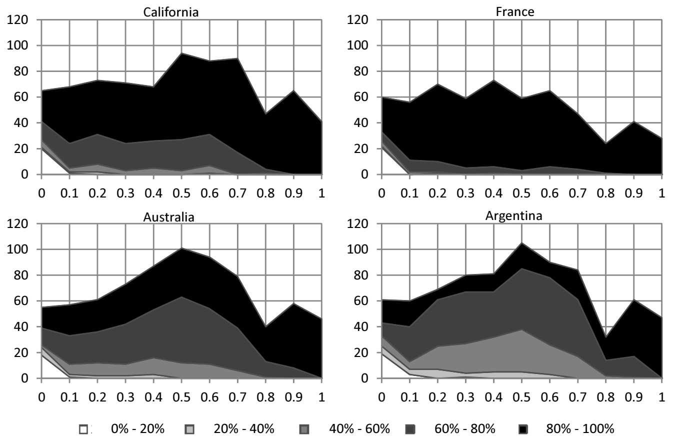

Fig. 6. dynamic file ratio histogram of every 0.1 in each host (vertical axis: number of web pages which have each ratio of dynamic files, horizontal axis: ratio of dynamic files, each area: percentage of onload time with cache in onload time without cache)

denotes the percentage of onload time with cache in onload time without cache. In Australia and Argentina, most web pages shortened their onload time to 80% or less by using the cache. In other hosts, a limited number of web pages shortened their onload time.

Figure 6 is a histogram of the dynamic file ratio of every 0.1 in each host. We divided the histogram into five areas in the same manner as in Figure 5. In Australia and Argentina, most web pages, of which 80% of files are dynamic, shortened their onload time to 80% or less. In other hosts, the number of web pages that shortened their onload times increased if 10% or less of their files were dynamic. However, this was observed in every host, and the reason for the increase in the number of short-onload-time web pages seems to be that there were a certain number of web pages whose files were all static; white area in Figure 6.

C. Discussion

As mentioned above, the proposed method shortened web pages ’ onload times in Australia and Argentina. The reason seems to be that the RTTs between the web servers and web browsers in Australia and Argentina are longer than those in other hosts. The difference in RTTs among hosts is shown in the “ Wait row ” of Table 1. In such long RTT hosts, time for downloading each file tends to be longer and the proposed method effectively shortened onload time of the web pages by reducing the number of files to be downloaded. However, in the hosts with sufficiently short RTTs, the proposed method had limited effectiveness in shortening the onload time of web pages. In other words, the proposed method is effective when the web browser displays a web page whose files are provided by long RTT web servers. For future work, we will implement a mechanism to determine the necessity of caching the rendering results in accordance with each web page’s RTT.

The proposed method is also effective in shortening the onload time of web pages whose percentages of dynamic files are less than 80% in Australia and Argentina and less than 10% in the other hosts. For this reason, we will implement a mechanism to determine the necessity of a cache with the percentage of dynamic files in web pages.

The reduction in short (in a few seconds) onload time without a cache tends to be large, as shown in Figure 5. The reason of this is that short-onload-time web pages tend to have many static files compared with long-onload-time web pages.

V. Related work

CDN services are used widely for acceleration web performance [7]. Using CDN, download time for each file in a web page is reduced. However, the waiting time in a web browser is reduced partially because there are still dependences between files and the browser waits for starting downloading.

Acceleration methods for web performance in a mobile environment by offloading rendering processing on servers or the cloud have been studied for a number of years. Kim et al. proposed pTHINC to show a web browser running in a server on a PDA screen using a thin-client technique [8]. Shen et al. proposed a web browser that quickly displays web pages containing dynamic objects, such as animations, even if the rendering process is done in a proxy server. The proxy server renders the dynamic and static portions separately and sends the rendering results of the dynamic portion and divided rendering results of the static portion to the web browser [9]. On the other hand, a web application framework was proposed that generates rendering results, such as a DOM tree in a web server, sends them to a web browser, and sends Ajax events to the web browser after sending the rendering results [10]. As mentioned above, there are various methods for offloading rendering processing and various types of divisions of roles between servers and browsers. Wang et al. argued about ways to divide roles between servers and browsers for efficient rendering processing [11]. Various studies on offloading rendering processing have been conducted and several products from such studies are being used [12].

However, there have been no studies on caching rendering results and reusing them.

VI. Conclusion

We proposed a method for web performance acceleration by reducing the rendering time using the cached rendering results. The rendering results are generated and cached by the in-network rendering function.

To evaluate the effectiveness of the proposed method, we calculated the rendering time by using the cached rendering results from actual rendering time measured using web browsers running on hosts worldwide. We found that the proposed method accelerates web performance in long RTT hosts and web pages. We also found that the proposed method accelerates web performance in such long RTT hosts and web pages of which 80% of files are dynamic. On the other hand, in short RTT hosts and web pages, the proposed method accelerates web performance of web pages of which less than 10% are dynamic.

We plan to implement mechanisms for determining the necessity of caching the rendering results in accordance with RTT and/or ratio of dynamic files. In addition, the proposed method classifies dynamic and static files by obtaining the differences between the rendering results generated from the same URL ' s web pages, but there seems to be errors in the classification. We plan to improve this classification by using a mechanism of considering cache control descriptions in HTTP headers. Finally, we will implement a prototype of the proposed method and measure web performance to evaluate the method.

REFERENCES

[1] S. Souders, *High Performance Web Sites Essential Knowledge for Front-End Engineers*. O'Reilly Media, 2007.

[2] I. Grigorik, *High Performance Browser Networking What every web developer should know about networking and web performance*. O'Reilly Media, 2013.

[3] "Http archive (har) format." [Online]. Available: https://dvcs.w3.org/hg/webperf/raw-file/tip/specs/HAR/Overview.html

[4] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "Demystifying page load performance with wprof," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL: USENIX, 2013, pp. 473–485.

[5] "Planetlab." [Online]. Available: https://www.planet-lab.org/

[6] "Alexa." [Online]. Available: http://www.alexa.com/

[7] E. Nygren, R. K. Sitaraman, and J. Sun, "The akamai network: A platform for high-performance internet applications," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, Aug. 2010.

[8] 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*, ser. WWW '06. New York, NY, USA: ACM, 2006, pp. 143–152.

[9] H. Shen, Z. Pan, H. Sun, Y. Lu, and S. Li, "A proxy-based mobile web browser," in *Proceedings of the International Conference on Multimedia*, ser. MM '10. New York, NY, USA: ACM, 2010, pp. 763–766.

[10] B. McDaniel and G. Back, "The cloudbrowser web application framework," in *Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity*, ser. SPLASH '12. New York, NY, USA: ACM, 2012, pp. 141–156.

[11] X. S. Wang, H. Shen, and D. Wetherall, "Accelerating the mobile web with selective offloading," in *Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing*, ser. MCC '13. New York, NY, USA: ACM, 2013, pp. 45–50.

[12] "What is amazon silk?" [Online]. Available: http://docs.aws.amazon.com/silk/latest/developerguide/introduction.html