IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received January 20, 2017, accepted February 22, 2017, date of publication March 29, 2017, date of current version May 17, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2689002

Reordering Webpage Objects for Optimizing Quality-of-Experience

WEIWANG LI[1], ZHIWEI ZHAO[1], GEYONG MIN[2], HANCONG DUAN[1], QIANG NI[3], AND ZIFEI ZHAO[1]

[1]College of Computer Science, University of Electronic Science and Technology of China, Chengdu 611731, China
[2]College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter EX4 4QF, U.K.
[3]Data Science Institute, Lancaster University, Lancaster LA1 4YW, U.K.

Corresponding author: Zhiwei Zhao (zzw@uestc.edu.cn)

This work was supported in part by the Fundamental Research Funds for the Central Universities under Grant ZYGX2016KYQD098, in part by the National Natural Science Foundation of China under Grant 61602095 and Grant 61303241, and in part by the EU FP7 CLIMBER Project under Grant PIRSES-GA-2012-318939.

ABSTRACT The quality of experience (QoE) perceived by users is a critical performance measure for Web browsing. "Above-The-Fold" (ATF) time has been recently recognized and widely used as a direct measure of user-end QoE by a number of studies. To reduce the ATF time, the existing works mainly focus on reducing the delay of networking. However, we observe that the webpage structures and content orders can also significantly affect the Web QoE. In this paper, we propose a novel optimization framework that reorders the webpage objects to minimize the user-end ATF time. Our core idea is to first identify the webpage objects that consume the ATF time but have no impact on the page experience and then change the positions of these objects to achieve the minimum ATF time. We implement this framework and evaluate its performance with popular websites. The results show that the ATF time is greatly reduced compared with the existing works, especially for complex webpages.

INDEX TERMS Web browsing, quality of experience (QoE), reordering.

I. INTRODUCTION

Webpages have become the standard for billions of users to access the Internet [11], [24]. The topic of Quality-of-Experience (QoE) perceived by web users has attracted much research attention in recent years [4], [12], [30]. As reported in many recent works, webpage loading delay has a strong correlation with web QoE: a lower delay leads to better web QoE [4], [7], [20]. Quantifying the QoE on webpage browsing seems quite complex because 1) the webpages nowadays have become highly complex which contain tens even hundreds of objects and Internet connections and 2) there is no clear signal/event that can be used to measure the user experience during the process of webpage loading and rendering. As a result, some new metrics are proposed to capture the actual QoE of web browsing, such as the Page Load Time (PLT) and Above-The-Fold (ATF) time [4], [6]. PLT denotes the duration from the time when the first request is sent to the time of the event *DOMContentloaded*. ATF time denotes the duration from the time the first request is sent to the time that the content above the first fold (i.e., the content that can be seen in the first-sight window of the browser) remains unchanged. Compared to PLT, ATF time is recognized as a more direct measure of the web browsing QoE [1], [5]. Therefore, we use the ATF time as the QoE metric throughout this paper.

Several works have been done to improve the web QoE. The first line of works focus on improving the network transfer for web browsing. For example, techniques such as DNS pre-resolution [29], TCP pre-connection [10] and the SPDY protocol [27] are utilized to reduce network delay at the application layer. Some works offload the computation tasks from the client browsers to the cloud-based proxy [2], [22], [25], [28], which is expected to reduce the delay experienced at the end users. The second line of works focus on reducing the loading delay. These works are designed to reduce the loading delay of web objects. Some other works [8], [13], [15] aim at reducing the computation delay of the javascript and css evaluation process. Although these works can improve the web QoE in some level, they often overlook the optimization space within webpage structures. The current webpage structures are often coded by the designers, which may not be well structured for end user QoE. Some objects that are placed in the beginning part of the webpage which may not contribute to the screen display but still incur considerable loading delay.

To address this issue, we investigate the problem of QoE optimization by modifying the webpages. The basic idea is to reorder the webpage objects in order to achieve the minimum ATF time, i.e., moving the objects that will not be shown in the first fold to the positions below the first fold, such that they are processed after the ATF time. To this end, there are two key challenges as follows: 1) Whether an object is related to the ATF display? It is quite difficult to run all the codes before reordering, thus we need to tell whether an object is display-related before actually loading it. 2) Whether an object is movable? Although some objects may not be related to the ATF display, they may be depended on by some objects that are related to the ATF display.

In this paper, we propose a novel QoE optimization framework that modifies the webpages to achieve the minimum ATF time. In the framework, a novel scheme based on Support Vector Machine (SVM) is proposed to estimate whether an object is related to the ATF display and ATF time, based on analysis of the HTML tags and code segments. Besides, we propose a scheme to judge the movability of an object based on dependency analysis similar to [26]. With the above modules, we are able to postpone the objects that have no or little impact with the ATF display below the fold and the ATF time can be reduced.

We implement the framework and evaluate it with the top 500 websites of China ranked in Alexa. The experimental results show that the reordering system can effectively speedup the webpage browsing by optimizing the object orders.

The major contributions of this paper are listed as follows.

1) We propose a optimization framework that modifies the webpages for better web QoE.
2) We propose a novel scheme that can estimate the impact of webpage objects without loading them.
3) We implement the proposed framework and evaluate its performance with real-world websites. The results show that our work outperforms the existing works in terms of the ATF time.

The rest of this paper is organized as follows. Section II presents the preliminaries and motivation of this work. Section III presents the main design. Section IV evaluates our work's performance with experiments. Section V summarizes the related work on web QoE optimization. Section VI concludes this work and discusses the future directions.

II. PRELIMINARIES AND MOTIVATION

In this section, we first introduce the detailed process of webpage loading and then discuss the motivation of this work with an illustrative example as shown in Figure 1.

A. PRELIMINARIES

Modern webpages are becoming more and more splendid, which contains HTML, CSS, JavaScript and multimedia objects, e.g., pictures, video, audio, etc. The structures are also becoming more and more complex, which makes webpage loading a non-trivial task. A typical web browser such as Google Chrome loads the webpages in the steps as shown in Figure 2: Object loading downloads the essential files contained in the webpage; HTML parsing reveals the structures and correspondence in the webpage; The evaluation of CSS and JavaScript is for the execution of scripts and styles; Finally, Rendering is done for the webpage display.

1) OBJECT LOADING

Object Loading fetches the objects requested by the tags in the HTML or user interactions. Objects are fetched from the cloud servers over the application layer protocol (http, https, SPDY, etc.) or from the cache of browser. It mainly consumes networking delay.

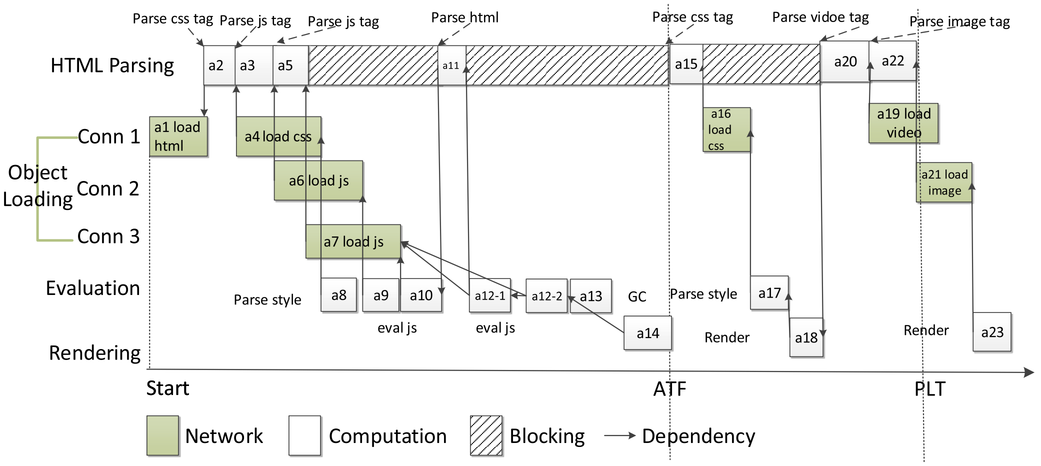


FIGURE 1. The working process of loading a given webpage.

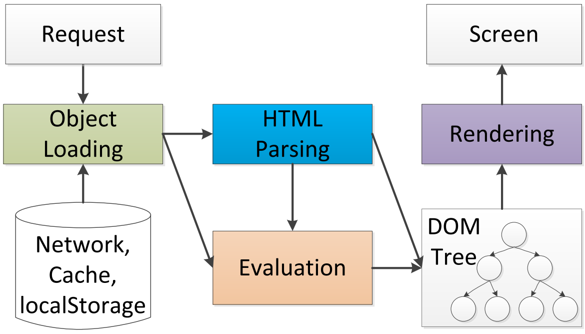


FIGURE 2. The general page loading process.

2) HTML PARSING

One of the main aims of HTML parser is to transform the HTML page to the document object model (DOM) Tree. DOM Tree is an intermediate representation of a webpage and provides a common interface for manipulation of the objects.

3) CSS AND JavaScript EVALUATION

CSS evaluation converts the CSS to the CSSOM Tree, which generates the style of the webpage. JavaScript can manipulate the DOM Tree. As both JavaScript Evaluation and HTML Parsing modify the DOM tree, HTML parsing is disabled during JavaScript evaluation to avoid conflicts in DOM modification.

4) RENDERING

Rendering is associated with two process-Layout and Paint. Layout transforms the DOM Tree to the layout tree and Paint converts this layout tree to pixel on the browser. We can see that HTML parsing, CSS/Javascript evaluation and rendering mainly consume computation delay. However, object loading and computational tasks are often executed in parallel. Besides, the dependency among objects also incur considerable delay in the entire webpage loading process.

B. MOTIVATION

We review the loading process of an example shown in Figure 1 and analyze the contributions of the objects to the loading time. The descriptions of all events are listed in Table 1. The loading process of the webpage is described as follows.

The main part of the figure shows the loading and rendering process. The green blocks denote the network delay of the object loading. The white blocks denote the computational delay during the parsing and rendering. The ''ax'' denotes the x-th event in the process. The arrowed lines denote the dependency. For example, the arrow from a12-1 to a7 means that a12-1 can be executed only when a7 is finished. We can see from the timeline, the content above the fold has reached the final state and remains unchanged when a17 is finished. All page contents are loaded and rendered when a23 is finished. From the above process it can be seen that, the execution of the first two .js file do not have visual impact on the ATF display but consumes computation time before the ATF (e.g., a7, a9, a10, a12).

TABLE 1. Event descriptions and the corresponding delay.

'a1'	25.6ms	downloads the .html file.
'a2'	1ms	parses the html till the 'main.css' tag.
'a3'	1ms	parses html till the 'pgturnEff.js' tag.
'a4'	27.2ms	downloads the 'main.css' file.
'a5'	1ms	parses html till the 'webcounter.js' tag.
'a6'	32.4ms	download the two .js files.
'a7'	35.2ms	download the two .js files.
'a8'	12.6ms	is the evaluation of 'main.css'.
'a9'	16.8ms	load 'pgturnEff.js' and 'webcounter.js'.
'a10'	18.2ms	load 'pgturnEff.js' and 'webcounter.js'.
'a11'	1ms	parses html file and push the tag elements into DOMTree.
'a12-1'	31.2ms	Executes the two js functions.
'a12-2'	34.5ms	Executes the two js functions.
'a13'	12.2ms	Garbage collection.
'a14'	10.5ms	rendering.
'a15'	1ms	parses the html.
'a16'	26.4ms	download .css file.
'a17'	29ms	evaluation for the .css file.
'a18'	13ms	rendering.
'a19'	62ms	load the video.
'a20'	1ms	parsing to the video tag.
'a21'	112ms	load the image.
'a22'	1ms	parsing to the image tag.
'a23'	16.7ms	rendering.

An intuitive idea is to execute the events related to these objects after the fold line, such that the processing delay will not be added to the ATF time. In this case, the ATF time can be greatly reduced when these four events are postponed after a17. However, we should also notice that although some objects (e.g., a4) are not relevant to the display above the fold, they are depended on by some objects that have impact on display above the fold. Therefore, such objects could not be moved below the fold.

III. DESIGN

A. OVERVIEW

Our design aims at reordering the webpage objects in order to achieve the minimum ATF time. The basic idea is to change the positions of the objects before the fold that have no impact on the display (termed as ''non-impactive objects''). Specifically, we would like to reorder them to the positions below the fold.

Figure 3 shows the proposed framework. Our reordering system acts as a proxy between the cloud web servers and the web publishers/developers. When the webpages are uploaded, our reordering system in the cloud changes the structures by reordering the objects and then publish the webpages for different devices. After that, the web user clients access the improved webpages with improved web QoE. There are three key building blocks in the reordering system.

1) Obtaining the fold. For different devices and different resolutions, the fold positions (the bottoms of the first-sight window) are different. Therefore, the first step is to obtain the fold position before finding the non-impactive objects.
2) Identifying the non-impactive objects. If all webpages are loaded and rendered at the proxy, the computational

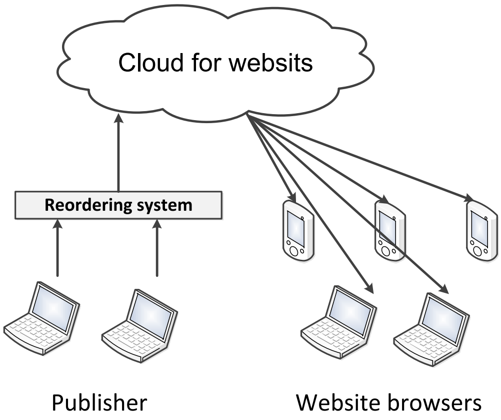


FIGURE 3. The proposed optimization framework.

overhead will be too large. Therefore, we would like to identify these objects based on the html files without loading the contained objects.

3) Judging whether the objects are movable. After we find out the non-impactive objects, we need to judge whether their positions can be changed by considering the dependency among objects.

After the above three steps, all movable non-impactive objects are moved below the fold to reduce the ATF time.

B. OBTAINING THE FOLD

ATF time is the time when the first-fold display no longer changes. The fold line is determined by: 1) the screen size and aspect ratio in the web client; 2) the resolution used in the web client. 3) the webpage content. The webpage content can be directly obtained from the uploaded files. To obtain the screen and resolution information, there are several different ways as follows. For mobile devices, the screen size and resolution can be inferred by the devices types. For other unpopular devices of which the screen and resolution parameters are unkonwn, the APIs provided by modern web browsers such as Chrome and Firefox can be utilized to directly read the information. With these information, we are able to calculate the fold line (≈browser's screen size).

With the fold line, it can be judged that whether an object is above the fold as follows: 1) the tags that placed above the fold; and 2) the related objects that manipulate these tags also belongs to the first fold. Only when the above two kinds of tags have been loaded and rendered, the first fold will remain unchanged. Although it is easy to find these ATF objects, it remains a challenge to identify the non-impactive ATF objects (or to identify the all impactive objects for the first fold).

As a result, although the first fold may not be a large part of the whole page, it may be fully loaded with much unnecessary delay because of that the related objects are not loaded/rendered in time. The more unrelated tags above the fold, the more space is left to be optimized for ATF time.

Next we explain how we identify the non-impactive objects above the fold.

C. IDENTIFYING THE NON-IMPACTIVE EVENTS

There are basically three types of non-impactive events. We review the example shown in Figure 1 to illustrate these events. First, the events that have no (either direct or indirect) impact on display. For example the garbage collections (GC) such as a13, which can be directly identified by the event identifiers. We note that there are also some non-impactive events that cannot be directly identified by the identifier (such as the JS event a12-1 which calculates a counter that will not be shown above the fold).

Second, the events that have no impact on the *ATF* display. These events are more difficult to be identified because although some events have no direct impact on the ATF display, they may manipulate the ATF tags and have indirect impact on the ATF display. Some events that have have no impact on display may be depended by the events related to the ATF display. We need to distinguish the non-impactive events from these indirect-impactive events. For example, some webpages need to read the users' IP addresses and the locations and show them in the first fold. The reading part may not be directly related to the display, but the display depends on the reading results. There are also some events that manipulate DOMTree but will not directly be displayed above the fold.

Third, we need to distinguish the events that seem related to display but have no actual impact on the webpage. For example, some javascript (js) and css code sections may be associated to a html tag that has been removed in the development of the webpage. In this case, these js and css codes become useless but may still remain in the html file and the webpage processing. Some other js and css files may be associated to the tags that is not shown in ATF part. Such cases often occur when the webpages are designed complex and the developers are not careful enough when adding/removing web tags.

To identify whether an event is non-impactive or not, there are several alternatives such as decision tree, Support Vector Machine (SVM), k-means, etc. Since the event information contains multiple dimensions such as the event name, URL, event size, etc., we use SVM in our work to identify the non-impactive events. The input of the SVM is the target features (described as follow) and the history data of non-impactive events. The output of the SVM is a judgment whether the events are non-impactive or not. In the very beginning we need to manually analyze the events and record the results as history training data for the SVM model.

D. FEATURES

Considering there are tremendous websites to be optimized when there are many users, we improve the processing efficiency of the identification, by reading only the html file as the feature extraction source, instead of all the files referenced in the html file. Though it is easy to identify certain events with clear identifiers, the challenge is to find the features for identifying the impact of .js and .css files contained in html.

1) FEATURES FOR .js FILES

For JS files referenced in URLs, the useful features include the JS file name, the domain name and the directory name in the URL. For JS code segments implemented in the html file, the useful features include not only the above features, but also the number of code lines, the function name in the code segments and the manipulated element name (e.g., the target element name in the "getElementById()" method). For example, the features for the JS insertion ("<link type='text/javascript' href='http://www.mobinets.org/misc/counter.js'>") include file name "counter.js", URL domain "mobinets.org" and URI "misc". Such information for each JS insertion can be collected and used for training the SVM model to judge whether the target JS is non-impactive. The training process will be presented in the following section.

2) FEATURES OF CSS

In general, .css files are related to the display. Therefore, our job is to judge whether the target .css file or code segment is related to the ATF tags. Similar to JS, the features for CSS also include file name, URL, directory, etc. For example, the features for the CSS insertion ("<link rel='stylesheet' type='text/css' href='http://www.mobinets.org/frame/main.css'>") include file name "main.css", URL domain "mobinets.org" and URI "frame". From the directory we can further obtain the domain name information and the file location by separating the URL with "/".

E. DATA PROCESSING

With the extracted features above, we need to transform the features to numerical input to the SVM model. The basic requirement is that the numerical results should be able to reflect the characteristic of the string features. For example, "frame" and "framework" are two close semantic terms. When transformed to numbers, the difference between numbers should be able to reflect the semantic distance of the features. We consider three alternative methods that can be used for converting string features into numerical features.

1) Sum of the ASCII values. With ASCII code we can directly convert the characters in each feature string to a series of numbers. Then the numbers are summed up as the numerical feature. For example, the "www.example.com" is converted to "1746" with ASCII code.
2) ASCII Vectors. Different from the above method, the ASCII values of each character in the string is saved in a vector to present the numerical features. For example, the "www.example.com" is converted to (119,119,119,46,109,121,101,120,97,109,112,108, 101, 46,99,111,109). Compared to the above method, the vector is a more precise representation of the features but will incur more computational overhead.
3) Minimum Edit Distance (MED) [19]. Another alternative is to calculate the minimum edit distance (MED) for each feature string. We use the average value of all the training set as the baseline for calculating the MED values. Each feature is represented by the vector of ASCII value difference between the feature and the average value. For example, given the vector of average values (119,119,119,46,111, 123,111,101,102,111,113, 101, 110, 80, 102,115,103), the "www.example.com" is converted to MED vector as (119-119, 119-119, 119-119, 46-46, 109-111, 121-123, 101-111, 120-101, 97-102, 109-111,112-113, 108-101,101-110, 46-80,99-102,111-115,109-103) = (0, 0, 0, 0, −2, −2, −10, 19, −15, −2, −1, 7, −9, −34, −3, −4, 6). We can see that the common part "www." are the same for different feature values and termed as four "0"s in the MED vector. We can also separate the URL with "." in order for the MED vectors to be more representative.

The numerical features are then fed into the SVM model. For the implementation of SVM, we follow the design proposed in [9]. Then we can judge whether a web object is non-impact or not. We will study the performance of the different converting methods in Section IV.

After obtaining the non-impactive events, the next step is to find out whether these events can be moved by analyzing the dependency. We follow the dependency analysis in [26]. If a non-impactive event is depended on by no other events or by events that are not related to the ATF tags, we mark this event as "movable". Next, we need to move these movable non-impactive events after the current fold line to reduce the ATF time.

F. REORDERING

The reordering module is to move the movable non-impactive events afterwards. Currently we move these events to random positions after the ATF. However when these events are moved, the fold line will change accordingly, which in turn impacts the identification of the non-impactive events. As a result, the fold line and the non-impactive events should also be updated before the next round of reordering.

Now we revisit the example shown in Figure 1. As shown in the code segment in Figure 4, the jsC1 function ("writeToThePage()") will write "Hello World" to the page (Figure 5) and the jsC2 function ("collectInput(3,4)") adds two input sources but displays nothing to the page. Apparently, jsC1 is impactive to the ATF display and jsC2 is non-impactive to the ATF display. From the codes and the settings of the client, we can obtain the fold line of the page (the bottom line of Figure 5). With this line, we construct the structure of the webpage as shown in Figure 4(a). We check the positions of the object one by one and obtain all objects that are located above the fold (included in the red line in Figure 4(a)). Then when jsC2 is identified as non-impactive, it is moved out of the ATF area. Then the ATF is updated as shown in Figure 4(b). The process continues

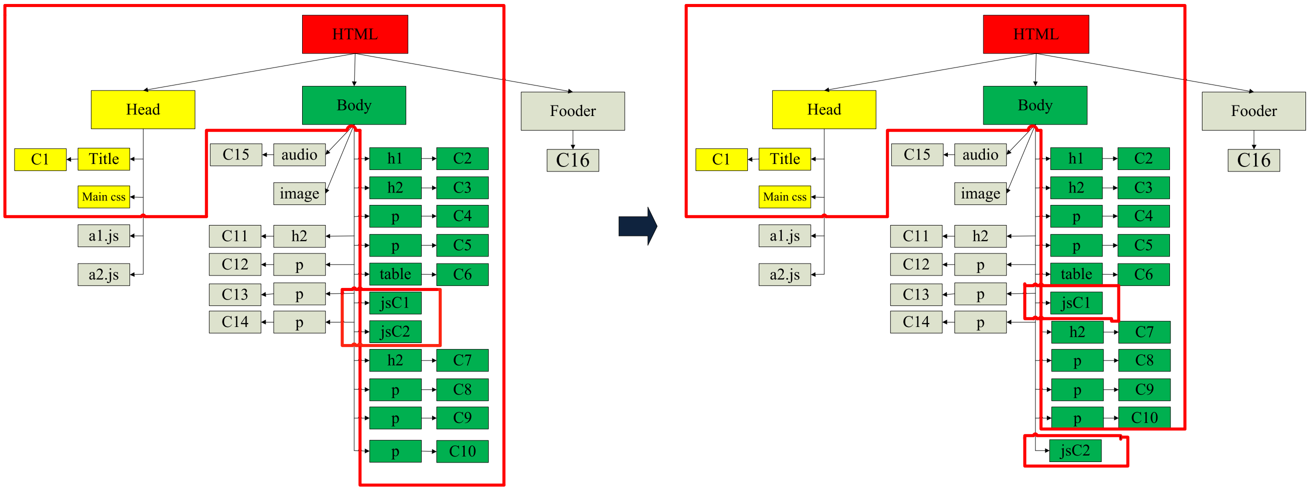
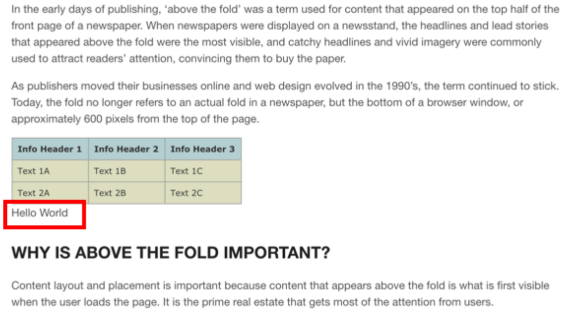


FIGURE 4. The original and reordered structure of the example in Figure 1.


In the early days of publishing, 'above the fold' was a term used for content that appeared on the top half of the front page of a newspaper. When newspapers were displayed on a newsstand, the headlines and lead stories that appeared above the fold were the most visible, and catchy headlines and vivid imagery were commonly used to attract readers' attention, convincing them to buy the paper.

As publishers moved their businesses online and web design evolved in the 1990's, the term continued to stick. Today, the fold no longer refers to an actual fold in a newspaper, but the bottom of a browser window, or approximately 600 pixels from the top of the page.

Info Header 1	Info Header 2	Info Header 3
Text 1A	Text 1B	Text 1C
Text 2A	Text 2B	Text 2C

Hello World

WHY IS ABOVE THE FOLD IMPORTANT?

Content layout and placement is important because content that appears above the fold is what is first visible when the user loads the page. It is the prime real estate that gets most of the attention from users.

FIGURE 5. Webpage display. The red rectangle denotes the output of the javascript.

until all non-impactive ATF events are moved afterwards (as shown in Figure 6). We can see that the ATF time of the same webpage if reduced from 45.2ms to 30.2ms with our reordering scheme. At the same time, the page load time is not increased.

G. DISCUSSION

It is possible that the identification of the non-impactive events is not accurate. If false negatives happen, the events will not be reordered. The ATF time will be larger than the minimum ATF time but can still be reduced compared to the original webpage ATF time. If false positives happen, the events that are impactive will be reordered. The ATF time will be enlarged because these events will be the new bottleneck of the ATF display. To reduce the negative impact of the false positive cases, we add a confidence threshold to the non-impactive event identification. Only when the confidence is larger than the threshold, the events can be identified as non-impactive.

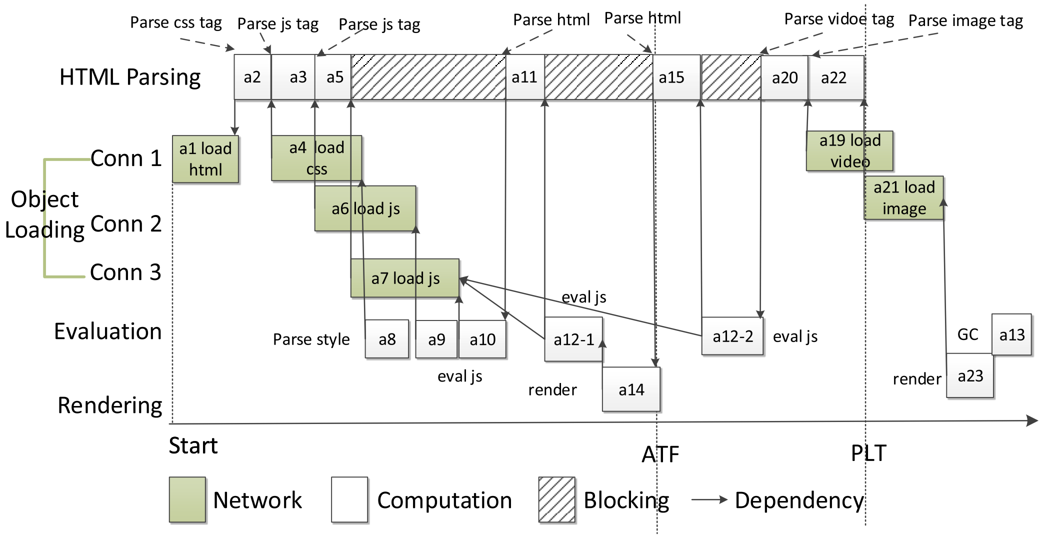


FIGURE 6. The optimized loading process of the webpage in Figure 1.

From the example in Figure 1, we can see that not all events can be moved afterwards because some events are not associated with specific webpage objects (Since only objects can be reordered in the webpage). For example, the GC events are not triggered by any objects. Reordering webpage objects cannot change the execution order of these events. Alternatively, the reordering of these events can be done at the client browser side. For other events that are associated with specific objects, we can directly move them afterwards the fold to reduce the ATF time.

It is also worth noting that there is optimization space for events reordering to further reduce the loading time of the entire webpage. We will leave the optimization of the moving strategy in our future work.

IV. IMPLEMENTATION AND EVALUATION

In this section, we present the implementation of the reordering and the evaluation results. As shown in Figure 3, the reordering system can be implemented either in the cloud side or in the publisher client side. In our experiment, we implement the system in the webpage publisher end (i.e., the webpages are first reordered before uploading to the cloud servers).

A. METHODOLOGY

We conduct the experiment on the Mac with the 2.8 GHz Intel Core i7 CPU, 16 GB 1600 MHz DDR3 memory and the Intel Iris Pro 1536 MB GPU in the Destop. We download the top 500 websites of China and save the website data locally. The data mainly includes the CSS files, JavaScripts files and the multimedia files. The webpages for training are first loaded and rendered locally using Google Chrome with the Chrome devtool (53.0.2785.143). The information includes the loading and rendering process of the webpage and the corresponding delay. Chrome devtool records the timestamps at the beginning and the end of each event executed during the page loading. The timestamps for four parameters are object loading, scripting, rendering and painting. We choose the top 300 and top 500 websites of China in the Alexa for training and testing our model, respectively. The different screen sizes and resolutions for different devices are considered as shown in Table 2. After the local reordering, the webpages are uploaded and then accessed by different devices. The network settings used in the experiment and the average round trip time (RTT) are also shown in Table 2.

We use all the three feature convert methods in our data processing and compare their performance.

B. EXPERIMENTAL RESULTS

Figure 7 shows the identification accuracy for non-impactive JS files with different converting methods. We can see that MED achieves the best accuracy. The reason is that: 1) its value can reflect more about the semantic difference between different feature values and 2) the non-impactive JS files with similar directory and file names to the non-impactive training data will be accurately identified. As the number of dimensions increases, the methods with ASCII vectors and the MED vectors increase. It is because that more fine-grained data achieves more accurate identification. Differently, the accuracy of identification with ASCII sum method decreases. The reason is that the sum of ASCII values cannot distinguish the cases with similar sum but with different values. For example, "mean.js" and "name.js" have the same ASCII sum value but the meanings are totally different.

TABLE 2. The network settings.

Notation	Meaning
good 2G	bandwidth 450kbps,150ms RTT
Regular 3G	bandwidth 750kbps,100ms RTT
Good 3G	bandwidth 1Mbps,40ms RTT
Regular 4G	bandwidth 4Mbps,20ms RTT
WiFi	bandwidth 30Mbps,2ms RTT

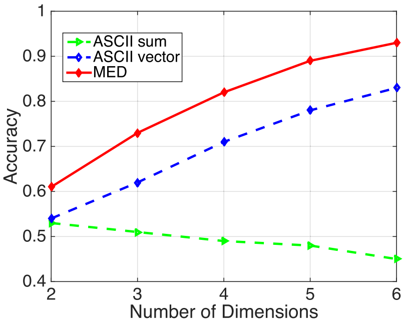


FIGURE 7. The identification accuracy of the JS files/segments.

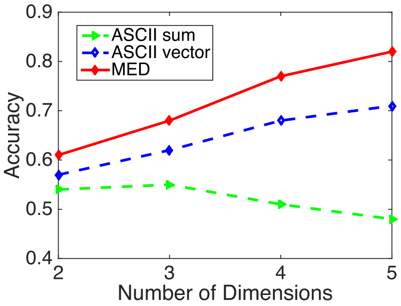


FIGURE 8. The prediction accuracy of the CSS evaluation's impact with different number of input dimensions.

In Figure 8, we compare the identification for non-impactive events with different converting methods. Similar to the JS identification results, MED vectors achieve the highest accuracy compared to the other two methods because it better represents the semantic difference and similarity among different feature samples. However, we notice that compared to the JS identification, the accuracy of CSS identification is lower. The reason is two fold. First, in many webpage development cases, the styles are often written in one single .css file. Therefore, the impactive and non-impactive

CSS segments often share the same file name and directories, which makes it difficult to distinguish. Second, there are often fewer CSS files/segments compared to JS files/segments. As a result, there are fewer CSS data for training the SVM model and the accuracy is reduced.

Next we evaluate the end-to-end improvement in ATF time. We choose the top 5 websites in China (in Alex ranking) as the target websites in the experiment. We compare the ATF time between the original webpages and the reordered webpages. Figure 9 shows the result. We can see that the improvement for baidu is the smallest and the improvement for sina is the highest (23%). By analyzing the webpages we infer that it is because the sina webpage is the most complex compared to the other four websites (containing the largest number of JS and CSS files/segments). From this case, we can also infer that the proposed work is more suitable for complex webpages because there may be more space for optimization.

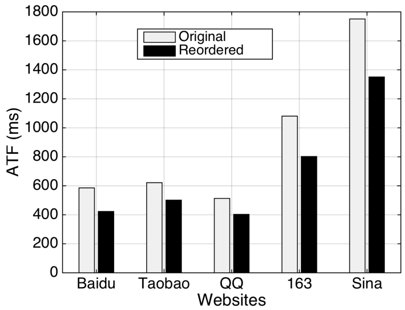


FIGURE 9. **The ATF time of the five target websites.**

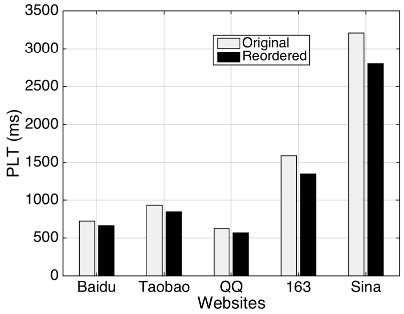


FIGURE 10. **The PLT of the five target websites.**

We also compare the entire page load time (PLT) between the original webpages and the reordered webpages. Figure 10 shows the result. (1) Compared to the ATF reduction, the PLT reduction is limited. The reason is that although we can optimize the first fold, all objects in the webpages will be processed eventually. Therefore, the PLT time of the original and reordered webpages is similar. (2) Compared to the original webpages, the PLT of reordered webpages is reduced. Although our reordering does not optimize PLT, the reordering may shorten the critical path in the webpage, which is also beneficial to reducing PLT. (3) sina achieves the largest reduction in PLT. Similarly to the evaluation results in ATF time, the reason is that sina page is the most complex webpage in the five websites and has the most space for optimization.

V. RELATED WORKS

The literature can be classified into two categories according to the optimization objectives: Works on reducing the network delay and works on reducing the processing delay. There are also works that focus on the cloud-based proxy, which deals with the tasks offloaded from the mobile devices.

A. REDUCING NETWORK DELAY

There are several works aiming at reducing the network delay. Some works exploit the preloading techniques such as [17]. The network delay can be reduced by establishing TCP pre-connection or TCP fast open technique [18]. Some works design new protocols for web browsing to speed up the browsing such as HTTP2 [23] and SPDY [27]. Local cache is also proposed and exploited for reducing the caching delay during the loading and rendering process in web browsing [3], [21].

B. REDUCING PROCESSING DELAY

Related measurement studies focus on improving page load computation. In Parallel Webpage Layout [15], a novel computing scheme is proposed to reduce the time of webpage layout by parallelizing the CSS evaluations. A similar technique called Parallel Web Scripting [13] is proposed to exploit parallelism between scripting and page layout under a new architecture. Parallel Web Browser [8] parallelizes preloading and preprocessing for web objects and can speed up the computation in sub-activities. The work in [14] reduces the page load time by parallelizing the loading and evaluation of JavaScript and CSS. In Silo [16] a new scheme is proposed to utilize the inline JavaScript and CSS files caching to reduce the round trip time.

1) REDUCING THE NETWORK/PROCESSING DELAY WITH CLOUD-BASED PROXY

For mobile devices with the computation and bandwidth limits, researchers have proposed to offload various types of functionality from client devices to cloud proxies. The work [25] proposed by Wang et al. exploits the mobile cloud computing to speed up the response and reduce the energy consumption. Flywheel [2] use the data compression proxy to reduce the transferred data size, which can greatly reduces the network delay especially in the non computation-intensive cases. Sophia [28] presents a proxy design to preprocess part of the web loading and rendering tasks, and parallelizes the CSS evaluation and JavaScript evaluation to reduce both network delay and the computation delay. FlexiWeb [22] utilizes network-aware image compression and adaptively decides whether to offload different web objects to meet the user deadline.

Our work differs from the above works that: 1) we aim at optimizing the webpage to minimize the end-to-end ATF time; 2) both network delay and the computational delay are implicitly considered; 3) the framework can be extended and implemented at the cloud side. It is also worth noting that our work is compatible to most of the existing works and can be simultaneously exploited for optimizing web QoE.

VI. CONCLUSION AND FUTURE DIRECTIONS

In this paper, we propose a novel optimization framework that modifies the webpage structures to minimize the user-end Above-The-Fold (ATF) time. We first identify the non-impactive objects and events that contribute to ATF time and then judge whether these objects and events can be moved considering the in-page dependency. The proposed work is implemented and evaluated with the top five websites in China according to Alex rank, the evaluation results show that the ATF time is greatly reduced with our work compared to the original webpages, especially for complex webpages.

Our future work will focus on the implementation of the work on the cloud side, where crowd sourcing can be used to further improve the accuracy for impact identification and reduce the ATF time.

REFERENCES

[1] *Reduce the Size of the Above-the-Fold Content*, accessed on Feb. 15, 2017. [Online]. Available: https://developers.google.com/speed/docs/insights/PrioritizeVisibleContent

[2] V. Agababov *et al.*, "Flywheel: Google's data compression proxy for the mobile Web," in *Proc. 12th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2015, pp. 367–380.

[3] S. W. Aiken and J. A. Saba, "High speed flexible slave interface for parallel common bus to local cache buffer," U.S. Patent 5 761 707, Jun. 2, 1998.

[4] A. Balachandran *et al.*, "Modeling Web quality-of-experience on cellular networks," in *Proc. 20th Annu. Int. Conf. Mobile Comput. Netw.*, 2014, pp. 213–224.

[5] E. Bocchi, L. De Cicco, and D. Rossi, "Measuring the quality of experience of Web users," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 4, pp. 8–13, 2016.

[6] J. Brutlag, Z. Abrams, and P. Meenan, "Above the fold time: Measuring Web page performance visually," in *Proc. Velocity, Web Perform. Oper. Conf.*, 2011, pp. 1–18.

[7] M. Butkiewicz, D. Wang, Z. Wu, H. V. Madhyastha, and V. Sekar, "Klotski: Reprioritizing Web content to improve user experience on mobile devices," in *Proc. NSDI*, 2015, pp. 439–453.

[8] C. Caşcaval *et al.*, "ZOOMM: A parallel Web browser engine for multicore mobile devices," *ACM SIGPLAN Notices*, vol. 48, no. 8, pp. 271–280, 2013.

[9] M. Chau and H. Chen, "A machine learning approach to Web page filtering using content and structure analysis," *Decision Support Syst.*, vol. 44, no. 2, pp. 482–494, 2008.

[10] Y. Deng and S. Manoharan, "Review and analysis of Web prefetching," in *Proc. IEEE Pacific Rim Conf. Commun., Comput. Signal Process. (PACRIM)*, Aug. 2015, pp. 40–45.

[11] L. Fan, L. Bonomi, L. Xiong, and V. Sunderam, "Monitoring Web browsing behavior with differential privacy," in *Proc. 23rd Int. Conf. World Wide Web*, 2014, pp. 177–188.

[12] D. Guse, S. Egger, A. Raake, and S. Möller, "Web-QOE under real-world distractions: Two test cases," in *Proc. 6th Int. Workshop Quality Multimedia Exper. (QoMEX)*, 2014, pp. 220–225.

[13] T. Hottelier, J. Ide, D. Kimelman, and R. Bodik, "Parallel Web scripting with reactive constraints," Dept. EECS, Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2010-16, 2010.

[14] C. G. Jones, R. Liu, L. Meyerovich, K. Asanovic, and R. Bodík, "Parallelizing the Web browser," in *Proc. 1st USENIX Workshop Hot Topics Parallelism*, 2009, pp. 1–6.

[15] L. A. Meyerovich and R. Bodik, "Fast and parallel webpage layout," in *Proc. 19th Int. Conf. World Wide Web*, 2010, pp. 711–720.

[16] J. Mickens, "Silo: Exploiting JavaScript and DOM storage for faster page loads," in *Proc. WebApps*, 2010, pp. 9–21.

[17] R. Mundwiler and M. Gaffin, "System and method for communicating pre-connect information in a digital communication system," U.S. Patent 6 178 173, Jan. 23, 2001.

[18] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan, "TCP fast open," in *Proc. 7th Conf. Emerg. Netw. Experim. Technol.*, 2011, p. 21.

[19] E. S. Ristad and P. N. Yianilos, "Learning string-edit distance," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 5, pp. 522–532, May 1998.

[20] A. Sackl, S. Egger, and R. Schatz, "The influence of network quality fluctuations on Web QOE," in *Proc. 6th Int. Workshop Quality Multimedia Exper. (QoMEX)*, 2014, pp. 123–128.

[21] J. Singh, B. T. Thio, C. W. Bhide, and W. R. Gray, "Installable performance accelerator for maintaining a local cache storing data residing on a server computer," U.S. Patent 5 805 809, Sep. 8, 1998.

[22] S. Singh, H. V. Madhyastha, S. V. Krishnamurthy, and R. Govindan, "FlexiWeb: Network-aware compaction for accelerating mobile Web transfers," in *Proc. 21st Annu. Int. Conf. Mobile Comput. Netw.*, 2015, pp. 604–616.

[23] D. Stenberg, "HTTP2 explained," *Comput. Commun. Rev.*, vol. 44, no. 3, pp. 120–128, 2014.

[24] P. Tsatsou, *Internet Studies: Past, Present and Future Directions*. Abingdon, U.K.: Routledge, 2016.

[25] H. Wang, J. Kong, Y. Guo, and X. Chen, "Mobile Web browser optimizations in the cloud era: A survey," in *Proc. IEEE 7th Int. Symp. Service Oriented Syst. Eng. (SOSE)*, Mar. 2013, pp. 527–536.

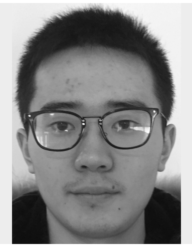
[26] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "Demystifying page load performance with WPROF," presented at the 10th USENIX Symp. Netw. Syst. Design Implement. (NSDI), 2013, pp. 473–485.

[27] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How speedy is SPDY?" in *Proc. 11th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2014, pp. 387–399.

[28] X. S. Wang, A. Krishnamurthy, and D. Wetherall, "Speeding up Web page loads with shandian," in *Proc. 13th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2016, pp. 109–122.

[29] R. S. Wilbourn, J. P. Wood, and R. T. Halley, "DNS resolution, policies, and views for large volume systems," U.S. Patent 8 707 429, Apr. 22, 2014.

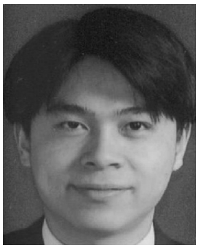
[30] M. Zhang, Z. Shen, X. Zhang, L. Luan, and Q. Ouyang, "Theoretical modelings for mobile Web service QoE assessment," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Mar. 2015, pp. 2026–2031.



WEIWANG LI received the B.S. degree from the College of Computer Science and Engineering, University of Electronic Science and Technology of China (UESTC). He is currently pursuing the M.S. degree with UESTC. His research interests focus on mobile computing.

ZHIWEI ZHAO received the Ph.D. degree with the College of Computer Science, Zhejiang University, in 2015. He is currently an Assistant Professor with the College of Computer Science and Engineering, University of Electronic Science and Technology of China. His research interests include wireless computing, heterogeneous wireless networks, protocol design, and network coding.

GEYONG MIN received the B.Sc. degree in computer science from the Huazhong University of Science and Technology, China, in 1995, and the Ph.D. degree in computing science from the University of Glasgow, U.K., in 2003. He is currently the Chair Professor and the Director of High-Performance Computing and Networking Research Group, University of Exeter, U.K. He joined the University of Bradford as a Lecturer in 2002. He became a Senior Lecturer in 2005 and a Reader in 2007, and was promoted to a Professor in computer science in 2012. His main research interests include next-generation Internet, wireless networks, mobile computing, cloud computing, big data, multimedia systems, information security, system modeling and performance optimization.



HANCONG DUAN received the B.S. degree in computer science from Southwest Jiaotong University, in 1995, and the M.E. degree in computer architecture and the Ph.D. degree in computer system architecture from the University of Electronic Science and Technology of China, in 2005 and 2007, respectively. He is currently a Professor of Computer Science with the University of Electronic Science and Technology of China. His current research interests include large-scale P2P content delivery network, distributed storage, and operating system.

QIANG NI received the B.Sc., M.Sc., and Ph.D. degrees from the Huazhong University of Science and Technology, Wuhan, China, all in engineering. He is currently a Professor and the Chair of Communications and Networking, School of Computing and Communications, Lancaster University, U.K. He has authored over 100 papers in engineering. His main research interests are wireless communications and networking. He is a Fellow of IET and was an IEEE 802.11 Wireless Standard Working Group Voting Member and a contributor to the IEEE wireless standards.

ZIFEI ZHAO received the B.S. degree from the College of Computer Science and Engineering, University of Electronic Science and Technology of China (UESTC). He is currently pursuing the M.S. degree with UESTC. His research interests focus on wireless and mobile computing.