HTML5 Web Worker Transparent Offloading Method for Web Applications

Zhao Wang
National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing 100190, China
e-mail: wangzhao@dsp.ac.cn

Haojiang Deng, Linlin Hu, Xiaoyong Zhu
National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences
Beijing 100190, China
e-mail: {denghj, hull, zhuxy}@dsp.ac.cn

***Abstract*—With the release of HTML5 standards, web applications have become more powerful, complicated and resource-hungry, whereas smart devices are in general resource-constrained. Computation offloading is one of the approaches used to increase application efficiency and decrease energy consumption of smart devices. In this paper, the offloading methods of HTML5 web worker in web applications are discussed, and a transparent offloading method of web worker is proposed to reduce execution time of web application and energy consumption of smart device. By rewrite of web worker implementation and modification of HTML5 websocket mechanism of web platform, web worker is offloaded to server side transparently. Common non-transparent web worker offloading method with JS framework has also been implemented. The experiments results show that the proposed web worker transparent offloading method achieves better performance improvements comparing to no offloading and non-transparent offloading respectively.**

Keywords-HTML5; web worker; websocket; computation offloading; transparent

I. INTRODUCTION

Computation Offloading has been considered to be an effective way to solve the performance problem of applications running on resource-constrained smart devices [1] [2] [3] [4]. By offloading some part of computation tasks, especially computation-intensive ones, applications can improve its performance and save resources of smart devices.

In HTML5 standards, there is a specification regarding web worker with which web applications can execute the parallelized workload like a thread. Previously before the introduction of web worker, the entire web logic is executed in a sequential way. If there is a complex computation task that is time-consuming, the UI thread can't response until the computation finishes. With web worker, parts of a web application can be executed concurrently separately from the main UI thread. Therefore, more complex/heavy web workload can be executed in web applications based on web worker. Fig. 1 shows the working mechanism of web worker. UI thread is main thread of web application and DOM is the structure of HTML web page in web application. Created by UI thread using a JS file, web worker can do some computation tasks but can't modify DOM structure of web application.

However, web workers may consume large amount of computing resources in some web applications such as Ray Tracing (http://nerget.com/rayjs-mt/rayjs.html) and Image Decrypt. Ray Tracing renders a 3D image using selected number of web workers to conduct matrix computation. Image Decrypt decrypts AES-encrypted image with web worker. These kinds of computation are high resource-consuming. Smart devices have relatively limited resources and this may degrade web applications' performance and user experience. A web worker transparent offloading method is proposed and implemented to solve this problem and experiments have also been conducted to verify the effectiveness of this method.

The rest of the paper is structured as follows. Section II introduces HTML5 websocket and related work. Section III describes our proposed web worker offloading method. Section IV presents the experiments and conclusions. Finally, summary is given in Section V.

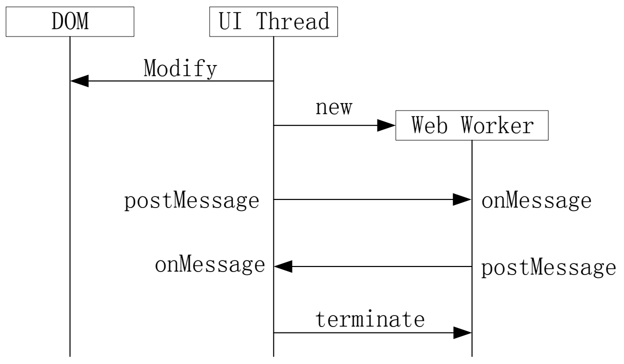

Figure 1. Web worker working mechanism.

II. HTML5 WEBSOCKET AND RELATED WORK

Apart from HTML5 web worker, there is another HTML5 specification concerning communication between web browser and server named HTML5 websocket, which is a full duplex communication protocol over a TCP connection. With websocket, a web application can establish connection with a server like a BSD socket interface. When connected to server with websocket, web application can send messages to server and receive messages from server by send and onmessage method respectively.

Nowadays there are two kinds of web worker offloading methods. One is transparent to web application and another is not transparent. Non-transparent web worker offloading is easy to implement and cross web platforms while need modification of web applications. On the contrary, transparent offloading does not change web applications but needs special implementation of web platform.

The general non-transparent web worker offloading methods rewrite standard web worker API with javascript and wrap it into a library. The web application which needs to offload web workers should be modified to include this library. The typical communication mechanism is HTML5 websocket. Web workers are created and running on server side, and the execution results are returned to client side through HTML5 websocket. This kind of method is proposed and implemented in [5] [6] [7] [8].

A-WWF is proposed in [9], which makes some modifications on the basis of WWF in [5]. The authors change WWF server side to be a manager to provide offloading service as well as relay the offloading requests of clients to other devices. The devices which can provide offloading service should register themselves to manager. WWF-D is proposed in [10], which uses HTML5 WebRTC communication mechanism instead of websocket. However, WebRTC is designed for real-time audio/video communication, which causes large communication cost and does not suit for web worker offloading.

HTML5 web worker transparent offloading has been outlined in [11]. The authors consider a system for augmenting mobile browser capability with cloud computation and storage resources. However, the authors do not provide the detailed design, nor the implementation and test of their solution. Another transparent web worker offloading system is proposed and implemented in [12]. By adding proxy module and offloading decision module in web platform, the authors offload web worker to a set of servers in cloud, and communication is implemented using ZeroMQ.

Besides of offloading, the relationship between web application performance and web worker numbers, CPU number, CPU architecture is studied in [13], which guides web application developers to create appropriate numbers of web worker to acquire best performance of web application.

Most of existing web worker offloading methods are non-transparent. To our best knowledge, [12] is the only implemented transparent offloading method. After ample research of current offloading methods, a new web worker offloading method is proposed. By rewrite of web worker implementation and a little modification of existing HTML5 websocket mechanism of web platform, web worker can be offloaded to server-side transparently. The proposed web worker offloading method has two advantages comparing to the offloading method in [12]. First, it can make the most of the existing communication mechanism of client web platform, which simplify the communication implementation between client web platform and server. Second, server side implementation becomes easier because of HTML5 websocket communication mechanism. It can even use the same server side as is used in general non-transparent offloading method.

III. Architecture of Web Worker Offloding Method

The running environment of web application is supported by web platform. Web platform provides corresponding implementation of the standard web API used by web application. Fig.2 shows the architecture of proposed web worker offloading method. In client side, web application uses standard web worker interface to create web worker and communicate to it. The platform implementation of standard web worker interfaces are rewritten, in which web worker creation request and communication messages are sent to server side based on modified HTML5 websocket. In server side, server main thread receives websocket messages from client, creates web worker thread, forwards communication messages to corresponding web worker, and returns results of web worker to client. The detailed design and implementation will be introduced in the following subsections.

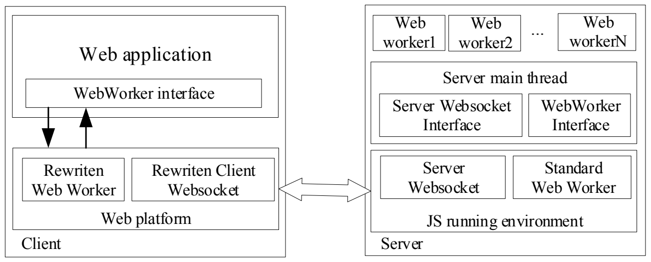

Figure 2. Architecture of web worker offloading method.

A. Implementation of Client

HTML5 standards provide JS interfaces for web application developers to create web application and the corresponding JS interfaces are implemented in web platform. To acquire transparency, the proposed method rewrites the corresponding platform implementation of web worker interfaces. In the creation process of web worker, it firstly establishes websocket connection with server side, and then sends necessary parameters to server side once websocket connection is established. When web application posts messages to web worker or terminates a web worker, corresponding messages are sent to server side based on the established websocket connection.

In web platform, the message types of web worker and websocket are totally different. To send web worker messages using websocket and acquire web worker messages from websocket, it is necessary to conduct transformation between web worker message and websocket message. Web worker message is first deserialized to acquire message contents and then the message contents is wrapped into a websocket message before sent to server side. A received websocket message should be converted to a web worker message before relaid to web worker message handler. For different functionalities, three types of web worker messages are defined, namely web worker creation message, communication message and web worker termination message, as is shown in Table I. Special tag is added in message contents to mark different type of messages.

Apart from web worker, it also needs a little modification of websocket. When a client websocket receives a message, it is necessary to determine if this is a web worker message or just an ordinary websocket message. For this purpose, the websocket message handler of client side checks the special tag of websocket message contents added by server side. If it is a web worker message, websocket handler transforms it into web worker message and relays it to web worker message handler.

TABLE I. WEB WORKER MESSAGE TYPE

message.type	Functionality
ESTABLISH	Client side establish a web worker
COMMUNICATE	Communication message between client and server
TERMINATE	Client side terminate a web worker

B. *Implementation of Server*

In server side, JS running environment plays similar role with web platform in client, supporting javascript running. Server main thread provides web worker offloading functionality. Server websocket listens client connection requests and receives messages from client when the connection is established. On receiving a message, server main thread first checks message type. If it is a web worker creation message, necessary parameters to create a web worker are acquired and a web worker is created. If it is a communication message, server main thread resolves message contents and relays it to corresponding web worker. Server main thread terminates corresponding web worker if it is a web worker termination message. It is worth noting that JS running environment supports standard HTML5 web worker. Web worker creation, communication and termination in server side all use standard web worker interfaces. When a web worker finishes its execution, it returns results to server main thread through web worker message. On receiving web worker message, server main thread resolves message contents and adds special tag to indicate that this is a web worker message and sends the messages to client side through websocket channel.

It is easy to learn from the web worker creation process that each client web worker creation request corresponds to a websocket connection, which corresponds to a web worker instance in server side. Therefore, when the server websocket receives a message from client, it is easy to relay the message to corresponding web worker. Because of the one-to-one correspondence of web worker and websocket connection, when a websocket is closed normally or because of an error, the corresponding web worker should be terminated to save resources even if client web application does not send termination message.

C. *Workflow*

Fig.3 shows the web worker offloading workflow between client and server.

1. When a new Worker is created by web application, the corresponding implementation in web platform first establishes a websocket connection with server and then sends url of the JS file that is used to create web worker.
2. When server websocket receives web worker creation message from client, it first checks message type, and then downloads the JS file, creates a web worker.
3. When web application posts message to web worker, the corresponding implementation in web platform transforms web worker message into a websocket message and sends it to server.
4. Server websocket receives client message, checks message type, extracts message contents and relays it to corresponding web worker.
5. Web worker receives message and returns results to server main thread through standard web worker message interface when it finishes execution.
6. Server main thread receives web worker message and relays it to client through websocket channel. Client websocket receives this message, checks that it is a web worker message and relays it to corresponding web worker message handler.

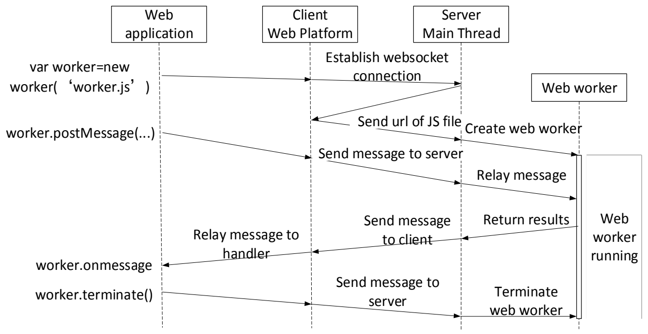

Figure 3. Web worker offloading workflow.

IV. EXPERIMENTS AND CONCLUSION

For comparison, the proposed web worker offloading method and the common non-transparent web worker offloading method both have been implemented. The hardware of client is raspberry Pi 3 with 1G memory, 64 bit Quad Core 1.2GHz processor. The web platform is Chromium browser [14]. Hardware of server is single Dell PowerEdge R730 server in Local Area Network. The JS running environment is Node.js which bases on V8 engine. It is worthwhile to note that the two kinds of methods share the same server side including server main thread code.

For experiments, Ray Tracing and Image Decrypt are used to verify the effectiveness and efficiency of proposed web worker offloading method. Ray Tracing renders two images of 400px*400px and 200px*200px respectively with different number of web workers under the conditions of no offloading, non-transparent offloading and proposed transparent offloading. Image Decrypt decrypts different number of images with web worker. One web worker is responsible for decrypting one image of 6K and 3K respectively under different conditions. From the start button clicked, the time cost by applications to finish the final display is calculated in the experiments. A noteworthy difference between the above two applications is that in Ray Tracing all web workers collaborate to finish a fixed big job, whereas in Image Decrypt one web worker is responsible for a fixed little job. As the increase of web worker number, the whole workload in Ray Tracing is changeless, but in Image Decrypt the whole workload is proportional to the number of workers. The experiments results are shown in Fig. 4 to Fig. 7, the data in which are average value of multiple experiments.

Fig. 4 and Fig. 5 show that comparing to no offloading, offloading web worker to powerful single server can improve web application performance by a large margin. But our proposed transparent offloading method shows better performance improvements, which reduces execution time by 65% and 36% comparing to no offloading and non-transparent offloading respectively. Fig. 6 and Fig. 7 also show better performance comparing to the other two situations. Comparing to Ray Tracing, Image Decrypt shows less performance improvements when offloading web worker to server side. This is because Image Decrypt has to exchange larger data with server side, which increases the communication cost. Similar to conclusion concluded in [15], the web worker which has large amount computation but little data exchange with web application is most suitable for offloading.

There is an interesting phenomenon that web application Ray tracing achieves best performance when the number of web workers are 4 under the condition of no offloading. This phenomenon is consistent with the conclusion in [13]. This is because raspberry Pi 3 is Quad Core. Web application can achieve best performance when the number of web worker are the same with CPU cores. When the number of web worker is bigger than 4, there are resource competitions between multiple web workers, which degrade the performance of web application.

V. SUMMARY

This paper proposed and implemented a transparent web worker offloading method based on HTML5 websocket. It was demonstrated that the method achieved better performance comparing to no offloading and common non-transparent offloading method. However, web worker offloading is not always beneficial. Sometimes, it's better to execute web worker locally. In the future, the offloading cost model and offloading decision strategies will be studied to measure offloading cost dynamically and determine whether it is advantageous to offload web worker to server side under different conditions.

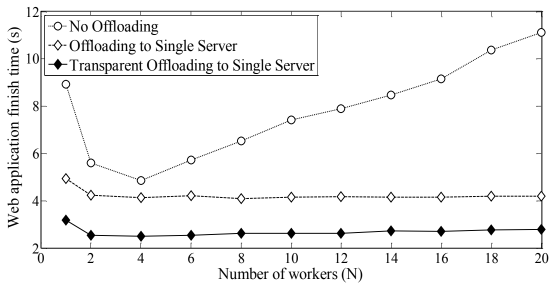

Figure 4. Experiment results of Ray Tracing (400px*400px).

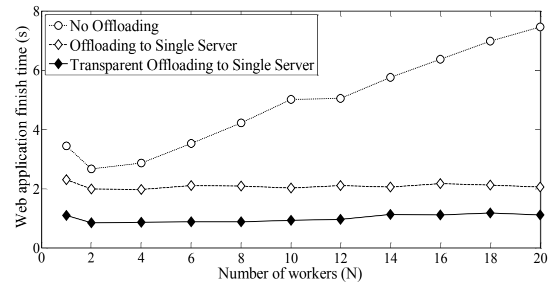

Figure 5. Experiment results of Ray Tracing (200px*200px).

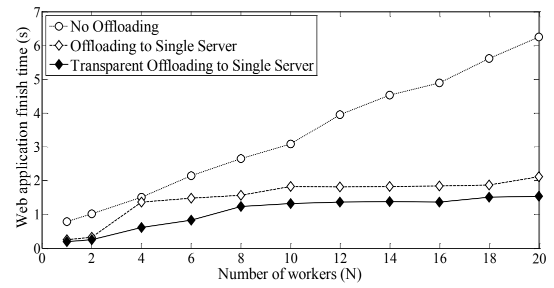

Figure 6. Experiment results of Image Decrypt (6K).

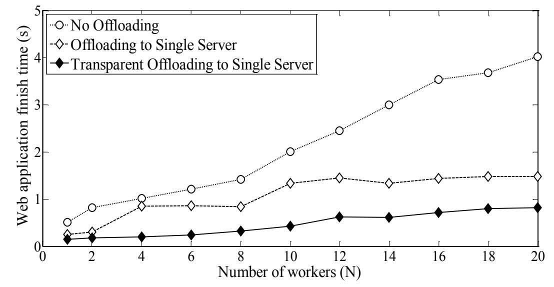

Figure 7. Experiment results of Image Decrypt (3K).

ACKNOWLEDGMENT

This work is partially supported by the Pioneering Action Initiative of Chinese Academy of Sciences (shuaixian xingdong jihua) (Project No. Y654101601). The authors would like to thank the anonymous referees for their comments and suggestions. Furthermore, the authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

References

[1] Xu, Chaoran, et al. "MOJA - Mobile offloading for JavaScript applications." Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies IET, 2014:59-63.

[2] Tseng, Tai Lun, C. H. Tu, and C. H. Tu. "Migratom.js: a JavaScript migration framework for distributed web computing and mobile devices." ACM Symposium on Applied Computing ACM, 2015:798-801.

[3] Fernando N, Loke S W, Rahayu W. Mobile cloud computing: A survey[J]. Future Generation Computer Systems, 2013, 29(1):84-106.

[4] Yu M, Huang G, Wang X, et al. JavaScript Offloading for Web Applications in Mobile-Cloud Computing[C]// IEEE International Conference on Mobile Services. IEEE, 2015:269-276.

[5] Hwang, Inchul, and J. Ham. "WWF: Web Application Workload Balancing Framework." International Conference on Advanced Information NETWORKING and Applications Workshops IEEE Computer Society, 2014:150-153.

[6] Hwang, Inchul, and J. Ham. "Cloud Offloading Method for Web Applications." IEEE International Conference on Mobile Cloud Computing, Services, and Engineering IEEE, 2014:246-247.

[7] Gong, Xiaoli, et al. "WWOF: An Energy Efficient Offloading Framework for Mobile Webpage." International Conference on Mobile and Ubiquitous Systems: Computing, NETWORKING and Services ACM, 2016:160-169.

[8] Kurumatani, S, M. Toyama, and E. Y. Chen. "Executing Client-Side Web Workers in the Cloud." IEEE, 2012:1-6.

[9] Hwang I. Adaptive Computational Workload Offloading Method for Web Applications[C]// International Conference on Computational Science and Its Applications. Springer, Cham, 2015:459-471.

[10] Hwang, Inchul. "Design and implementation of cloud offloading framework among devices for web applications." Consumer Communications and NETWORKING Conference IEEE, 2015.

[11] Zhang, Xinwen, et al. Elastic HTML5: Workload Offloading Using Cloud-Based Web Workers and Storages for Mobile Devices. Mobile Computing, Applications, and Services. Springer Berlin Heidelberg, 2012.

[12] Zbierski, Maciej, and P. Makosiej. "Bring the Cloud to Your Mobile: Transparent Offloading of HTML5 Web Workers." IEEE, International Conference on Cloud Computing Technology and Science IEEE, 2015:198-203.

[13] Verdu J, Pajuelo A. Performance Scalability Analysis of JavaScript Applications with Web Workers[M]. IEEE Computer Society, 2016.

[14] Wang S, Sun P, Guo Z, et al. Design and Implementation of Embedded Web APP Engine[J]. Journal of Network New Media, 2016.

[15] Kumar K, Liu J, Lu Y H, et al. A Survey of Computation Offloading for Mobile Systems[J]. Mobile Networks & Applications, 2013, 18(1):129-140.