J Inf Process Syst, Vol.13, No.4, pp.731~751, August 2017
https://doi.org/10.3745/JIPS.04.0037

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

Cloud Computing to Improve JavaScript Processing Efficiency of Mobile Applications

Daewon Kim*

Abstract
The burgeoning distribution of smartphone web applications based on various mobile environments is increasingly focusing on the performance of mobile applications implemented by JavaScript and HTML5 (Hyper Text Markup Language 5). If application software has a simple functional processing structure, then the problem is benign. However, browser loads are becoming more burdensome as the amount of JavaScript processing continues to increase. Processing time and capacity of the JavaScript in current mobile browsers are limited. As a solution, the Web Worker is designed to implement multi-threading. However, it cannot guarantee the computing ability as a native application on mobile devices, and is not sufficient to improve processing speed. The method proposed in this research overcomes the limitation of resources as a mobile client and guarantees performance by native application software by providing high computing service. It shifts the JavaScript process of a mobile device on to a cloud-based computer server. A performance evaluation experiment revealed the proposed algorithm to be up to 6 times faster in computing speed compared to the existing mobile browser's JavaScript process, and 3 to 6 times faster than Web Worker. In addition, memory usage was also less than the existing technology.

Keywords
Cloud, HTML5, JavaScript, Mobile

1. Introduction

A variety of mobile services is emerging in response to the explosive global popularity of smartphones. The mobile application market has evolved from a closed pattern of operations run by existing telecommunications operators to an open pattern that supports entrepreneurial general and web-based applications. This change has laid the groundwork for the rapid expansion of mobile application development and market size, and increased accessibility to information and services offered through various mobile applications. The current smartphone application environments are divided into several markets. The Wholesale Application Community (WAC) joined the Group Special Mobile Association (GSMA) in July 2012; the latter association targets the number of mobile phone users of the participating telecom operators, accounting for two-thirds of the estimated 3 billion global mobile phone users, with future penetration into this market likely. The WAC provides various libraries and standard documents useful in the development of web applications at the same level as native

Manuscript received November 1, 2016; first revision January 18, 2017; accepted March 4, 2017.
Corresponding Author: Daewon Kim (drdwkim@dku.edu)
* Dept. of Applied Computer Engineering, Dankook University, Yongin, Korea (drdwkim@dku.edu)

applications by allowing the use of powerful HTML5 [1,2] user interface functions and mobile resources.

Those applications use JavaScript [3,4] to access the internal resources from mobile devices. JavaScript is a language that is significant in configuring the web service client environment and is processed by the interpreter method. Therefore, when simultaneously queried for multiple operations, JavaScript stops while the web browser runs to perform processing. Another problem is the relatively slower processing speed of JavaScript engines compared to native applications when performing complicated processes or executing an algorithm with many operations. One alternative to lessen these limitations is to use just-in-time (JIT) [5] compilers to improve JavaScript engine performance speed. However, disadvantages of JIT include slow response due to limited mobile resources, increased memory usage [6], and incapability for the mobile environment. Efforts to overcome these hindrances are addressing the reuse of compiled codes [7]. Another alternative is a method that supports JavaScript multi-thread programming. This method prevents execution speed- and screen-related stoppage by performing multi-thread parallel processing. Document Object Model (DOM) Workers [8,9] provide multi-thread functions, but they are unable to access DOM directly and do not share the name space. To resolve these issues, HTML5 provides Web Worker functions [10]. The use of Web Workers enables much of the run-time to be separated and processed in the background, which makes prompt processing possible without freezing of the screen. In addition, it is possible to access the DOM and use the namespace through the message during execution. A disadvantage of this approach is the need for more run-time than native applications, despite powerful Web Worker functions, due to a restricted mobile environment. This paper introduces a novel cloud framework to improve the operating performance of web-based applications in the restricted mobile handset environment. In the proposed method, JavaScript used in the existing client is instead performed in the server with relatively high hardware specifications, whose results are taken as a response and processed. Performance assessment revealed 3 to 6 times faster performance improvement compared to performance when locally executed. Also, memory usage was reduced by transferring the execution code to the server. Section 2 of this paper outlines the existing methods for improving JavaScript performances. Section 3 describes the structure as well as technical performance procedures. Section 4 presents performance assessment results in comparison with existing methods. Finally, Section 5 provides a discussion, conclusions and future research directions.

2. Relevant Research

2.1 JIT Compiler

JavaScript is a scripting language designed on the condition that it is executed by an interpreter. JavaScript is commonly used to implement dynamic elements of web pages showing responses according to what the user types or time in the web browser, such as an auto complete function or real-time search ranking display. Beginning recently, to provide a variety of services, many web pages are being implemented using a JavaScript program requiring a large number of operations. However, a problem arises concerning the user response time due to notable lag in speed when the code is converted in the engine into an abstract syntax tree or intermediate code before being performed by the interpreter. To improve this, some JavaScript engines use a JIT compiler during the execution to directly

perform machine code that translates JavaScript code or its intermediate code using a JIT compiler. Of JavaScript engines, JIT compilers are used by V8 [11] and Mozilla's open-source browser, TraceMonkey [12]. TraceMonkey executes a program in the interpreter by interpreting SpiderMonkey [13] JavaScript code into intermediate code. As an alternative to this limited performance, TraceMonkey implemented a JavaScript engine that improves performance by tracing a series of intermediate codes operated in the interpreter to detect a repeated statement, and interpreting it into machine code by a JIT compiler before execution. Fig. 1 presents the machine-code generation process in TraceMonkey using a JIT compiler, with panels 'A'–'D' denoting JavaScript functions or codes.

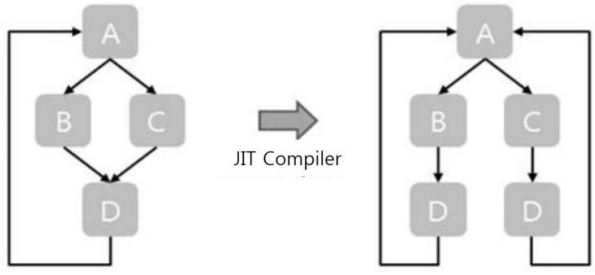


Fig. 1. Mechanical code generation process using the JIT compiler in TraceMonkey. (a) JavaScript code and (b) complied code.

When there is a branch statement repeatedly evident in JavaScript code in A of Fig. 1, the JIT compiler in TraceMonkey determines this as a frequently-executed path when performed in the interpreter [14,15]. Hot-Path is used to expand the existing code ('B' in Fig. 1) beginning where the repeated part begins, according to the path obtained when generating the initial code [16]. The code is expanded depending on the flow of the program execution therefore duplicate code is generated ('D' in Fig. 1). Processing speeds can be improved because the process of performing the crucial aspect of execution time of a program not by an interpreting method but by machine code following the compiling. This approach has an inherent disadvantage in that when the execution path changes by a conditional statement in the repeat statement or overlapped repeat statement, compiling is completed with the addition of a new path, which leads to the increase of run-time. The V8 JavaScript engine translates firsthand and executes all functions called from JavaScript code into machine code using a JIT compiler without going through an intermediate code [17]. This has the advantage of enabling all codes to be promptly implemented with machine code as an analyzer is not needed for such implementations. It also features the manufacture of machine codes dependent on library functions [18]. These JavaScript engines currently being used in various commercial terminal environments have the disadvantage of being unable to provide a processing speed that is as fast as the desktop browser due to limited environment of the mobile ecosystem [19,20].

2.2 JavaScript Multi-thread

Since JavaScript and DOM in the existing commercial browser operate on a single thread, they perform only one single job in a specific period of time. One disadvantage of such a processing structure is that its speed is very slow compared to other programs that process concurrently. To resolve

this issue, methods that have been implemented include DOM Workers and HTML5 Web Worker [21]. DOM Workers were mounted on the open-source browser early in the browser's development state provided by the plug-in and used for experimental purposes. Workers themselves have no access to DOM and are programmed to process simple operations separately. For example, with respect to processing many repeat statements or recursive functions, they are programmed to be separated in *js* files and processed. This method influenced even HTML5 for which standards were established and consequently incorporated Web Worker as standards [22]. Web Worker currently being supported by HTML5 can perform multi-processing and enable access to DOM using *postMessage*. This function has the advantage of increasing processing speed by providing concurrency more than what is processed in a single thread [23,24]. Fig. 2 depicts how HTML5 Web Workers run. When running an instance to use Worker, the JavaScript code in Worker.js is executed in a separate thread. Then, to communicate with the main thread executing HTML5, *postMessage* is used to deliver a message and the delivered message is processed through the message. Moreover, as background operation is possible, many mobile resources are likely to be utilized in accompanying smartphone applications [25,26].

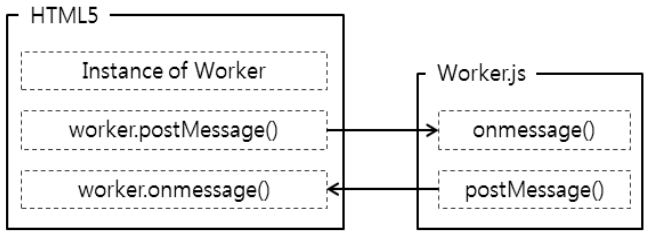


Fig. 2. Method of operation for Web Worker in HTML5.

Web Workers make JavaScript processing effective in terms of function, but the results of the test conducted in this research revealed that the browser on a desktop PC displays lower performance due to limitations in mobile resources [27]. A cloud framework is proposed to improve the functions of smart terminal application programs that demand complicated implementations in limited mobile environment.

3. Cloud Framework to Improve the Processing of JavaScript with Mobile Applications

3.1 Cloud Framework

The limited resource of the mobile environment presents performance limitations for a basic built-in JavaScript engine in a browser. Hardware performance of cellular phone terminals is continuously improving with a similar level of performance to a desktop PC, yet low-end terminals exist. To solve this problem, the proposed framework that provides services close to an advanced level of hardware, despite a low level of hardware resources, is introduced. This was designed for performance improvements of GSMA-based applications that are performed with JavaScript with the use of the principles of cloud computing [28]. The GSMA applications are divided into two groups. One is local

web application that is directly installed on mobile terminals and the other is remote web application which provides application services using web servers from the internet. In case of remote web applications, the server's abundant computing resources could be utilized, however, for the case of local web applications, the client's limited computing power is the only method of operation. In a multiprocessing environment, processing time measurements are affected by internal job processing situations. This allows the user to make a choice whether to run locally or online with the proposed cloud system so that they can decide whether to use it or not before main processing. Therefore, it is recommended that we analyze the data based on the existing log data about the processing time and use it when the processing time of the proposed cloud system is faster than local method. The log data also records and manages the maximum utilization of the CPU and memory as a reference indicator, including processing time, to determine whether the proposed system should run or not. Thus, we use preliminary measurements of processing time, average CPU and memory utilization to determine the use of the proposed cloud system. The system in this paper is basically for the JavaScript performance improvements of local applications software in mobile terminals. For instance, the proposed cloud framework method can be used for a local web application that can change automatically friends' pictures registered in terminal's phonebook by face recognition when captured through a phone camera. This picture-changing scenario is in the use case. The cloud framework system in this research can be managed and charged by a program developer directly or open to the public at a low price or for free. The GSMA related program developers in a small business can use this system to save expenses and enjoy the fast and strong computing powers. Local web applications could be run by using the only server's computing resources without recording or remaining data in the server. The cloud framework provides a function of reducing the amount of operations of JavaScript in the mobile terminal by partly transferring the JavaScript processing that demands a lot of run-time from the client and calling its processing results to the client before use.

Use case of the proposed cloud framework system
Step 1: An user runs a GSMA based local web application program.
Step 2: The user chooses pictures of friends saved on the terminal and inputs each of their names.
Step 3: The user tries to change a friend's picture registered on a phonebook of the terminal.
Step 4: The application program recognizes the picture locally and estimates processing time and usage of CPU and memory to edit.
Step 5: In order to decide whether to use the proposed system or not, the network delay and executing time between the client and server and utilization of CPU and memory need to be expected in advance.
Step 6-A: If the expected running time and the usage of CPU and memory are greater than the local processing amounts, then the case is dealt in local and moves to Step 8.
Step 6-B: If the expected running time and the usage of CPU and memory are less than the local processing amounts, then the chosen pictures and name information is transferred to the system's server.
Step 7: The server edits received pictures and sends back to the terminal. Remaining data in the server need to be deleted.
Step 8: The application program changes each pictures of friends in the phonebook correspondingly and finishes the processing.

Fig. 3 is a concept drawing of the proposed system, in which the mobile client delivers the JavaScript requiring much time during the execution to one of the cloud-based, process-enabled servers to process the server. In Fig. 3, the proposed cloud system has a merit that users are able to utilize high-fidelity server from a remote location using the JavaScript language in clients unlike other cloud systems. In

case of some other cloud services, it is only possible for users to use pre-specified programs, and there are shortcomings of sharing resources between servers of many companies. The proposed system in Fig. 3 is a complementary framework for those drawbacks.

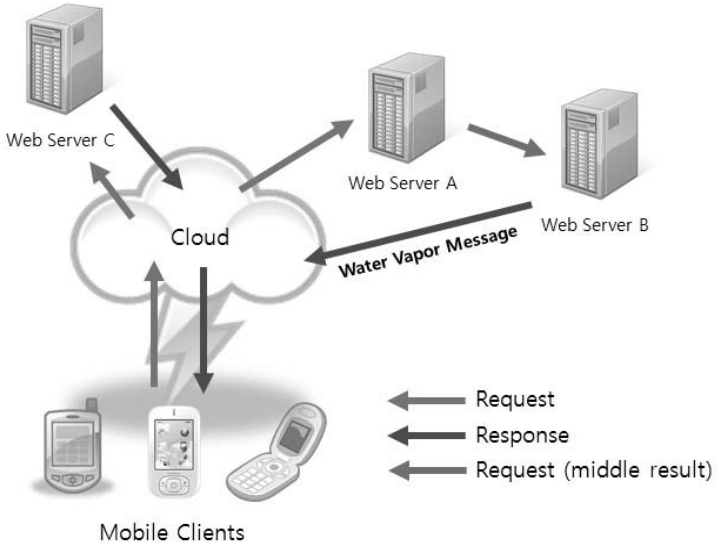


Fig. 3. Concept of the proposed cloud system and operating structure.

The cloud computing system used in this research is the type of IaaS (infrastructure as a service) which lends hardware resources so that the web applications could utilize the server's abundant computing powers. If there is a JavaScript code that requires high quality computing resources in a local web application, then the application developer could improve the local web application's performances using the cloud framework server system described in this study. There are two ways to process JavaScript. The first method is a single processing system. In Fig. 3, the client configures a JavaScript function to execute or the file containing its contents in format of water vapor (WV) and transmits it to the server. The WV protocol, a message with a JavaScript Object Notation (JSON) structure, allows the web server C in Fig. 3 to receive and process it [29], whose results are converted again into a WV form and transferred to the corresponding client. Second is a server relay method; when the WV message transmitted by the client has several server addresses recorded, the record is viewed and processed while travelling between the servers. In the process, a client requests web server A to process JavaScript, and delivers its results to web server B, and re-transmits the final results to the client. Fig. 4 systematically depicts the framework structure. The system in Fig. 4 makes use of a method of performance using JavaScript in the browser, so it can be included in the GSMA framework as well as in the application being performed to be processed. The system basically operates by communication between browser where applications run and server. Although services are provided focusing on native application software in existing cloud system, the proposed system mainly supports web-based applications, specifically for mobile web-applications that are much restricted within low performance and hardware capacity. For communication in JavaScript, Ajax is used for an identical domain, whereas WebSocket for a different domain. Now that GSMA creates applications with HTML5 as standards, the proposed system carries out communication with the use of HTML5 WebSocket being also accessible to other

domains, instead of Ajax. The HTTP Communication portion in Fig. 4 is a module used for communication with the server. To process the client-server request, a WV message format is used. As the process code comprised of JavaScript is inserted into what the client makes request to the server, the requested server transmits the message in a WV format to process this. Therefore, the WV message controller to create or delete such a WV message exists.

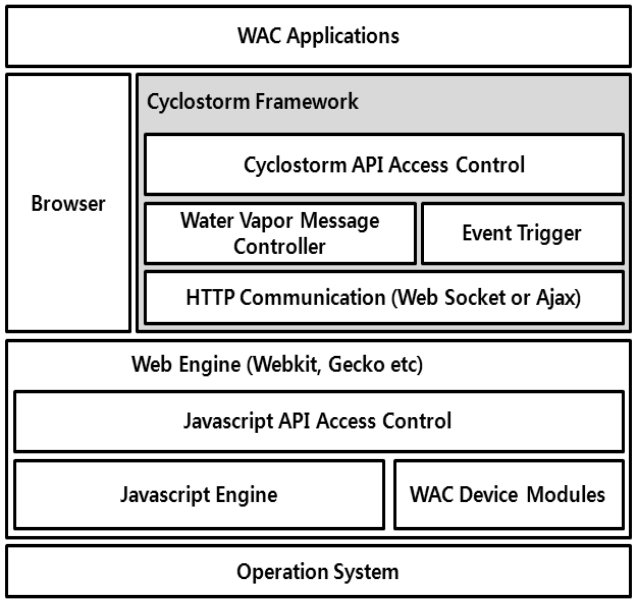


Fig. 4. The proposed cloud framework structure (named Cyclostorm).

Event trigger plays a part in calling the previously defined function in the code to correspond to the event taking place at the time of delivering and receiving a WV message. In addition, the system's API access control represents an interface and the gathering of interfaces provided by API composed of JavaScript. Ultimately, the system is provided in the form of a library so that JavaScript can be executed in the GSMA applications where JavaScript operates.

3.2 WV Message

WV message is a kind of executable message that has the function codes to be executed in the server and the values of factors that enter each parameter. The WV message is transmitted using the HTTP technology and it includes data for executions and JavaScript codes that run on a cloud server. The server in receipt of the WV message from the client processes the JavaScript included in it and transmits its results to the client, when the result values are included in the WV message and transmitted.

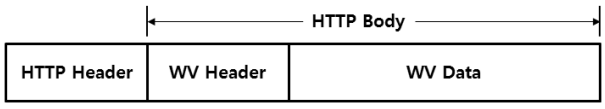


Fig. 5. Structure of a WV message.

Fig. 5 shows a structure of the WV message. From Fig. 5, the WV header includes request and response information and execution codes. The WV data contain information in the type of key-values for running JavaScripts. Table 1 summarizes the system of requested WV message's header format. In Table 1, Version indicates the version of a WV message and becomes an identification value for identification purposes when making multiple requests to the single server or several random servers.

Script indicates the type of the code executed in the server, and hosts denote an address to which the server will be connected. Since there are some cases where requests are made for processing through several servers, an array is used when displaying the address of a pass-through server.

Table 1. Water vapor message request header format

Name	Contents	Example (JSON data type)
Version	Water vapor message version	{"version":"1.0"}
Identification	Recognition value for 1:N request	{"id":"2100"}
Script	Script type to run in server	{"script":{"language":"JavaScript", "type":"text/JavaScript"}}
Hosts	Server address for 1:N connection	{"hosts":{"a.com", "b.com"}}
Flag	Request type value	{"flag", "start"}
Content	Function code with JavaScript	{"content":{"args":[100, ,"hello"], "func":"functon a(arg1, arg2) { ..... }"};

Table 2. Water vapor message request flags

Name	Content
Start	Code-run start request to server
Resume	Resume request to paused server
Suspend	Pause request of code-run
Stop	Stop request of code-run to server
Post	Asynchronous message processing request to executing server

Flag is a value used to make a request to the server, and its kinds are listed in Table 2. The Flags in Table 2 is similar to the name of a function that operates threads on the local. Start denotes the implementation of the code delivered to the server, whereas resume and suspend denote pausing or restarting the code being currently executed in the server. Stop means to request for the server in process to close. Post is used when making asynchronous requests to bring current in-progress situations or internal values to the server, or when a new value is allocated. Finally, in the Content seen in Table 1, actual function codes to be performed in the server and the values of elements to be used in parameters are stored in JSON format. Table 3 describes a function performing factorial operation within the Content as an example of JavaScript codes.

Table 3. Example of a content

Factorial computation function

{ "args" : ["10"], // first parameter factor value
"func" : " // factorial computation function code
function factorial(*n*) {
if (*n* > 1) { return *n**factorial(*n*-1); }
else { return 1; } }" }

The "args" in Table 3 represents a parameter value when performing a function defined in "func". This is composed of arrays and set as the values of parameters and elements when performing functions in orders of 0, 1, and 2. If any contents in the WV data are used as parameter values, then the key-value of the WV message is applied. The "func" term indicates a function code to be executed. The value in the 0th "args" array, when the server runs "func", is allocated to the value n out of 'function factorial(n)' in and implemented. When the requested function code is performed and closed, the server notifies the client of its results. Table 4 shows the response format of the WV message used during this time.

Version in Table 4 indicates the version of a WV message, and Identification represents the identifying value for the requested message. Script means a kind of script that can execute the value saved in Content, while Hosts is the address array of the server sending a request for response, where the array number 0 in the front part represents the address of the server issuing a response, and other addresses mean the addresses of the server with remaining requests.

Table 4. Water vapor message response format

Name	Contents	Example (JSON data type)
Version	Water vapor message version	{"version":"1.0"}
Identification	Response message version from a client	{"id":"2100"}
Script	Script type to run in a client	{"script":{"language":"JavaScript", "type":"text/JavaScript"}}
Hosts	Server address sent the current response message	{"hosts":{"a.com", "b.com"}}
Flag	Response type value	{"flag", "stop"}
Content	Final results after execution	{"content":{"data":"1234"}};

Table 5. Water vapor message response flags

Name	Content
Success	Message of successful execution status for transferred request
Error	Message of server execution error
Redirect	Message of request transfer to the next server
Post	Transmission of execution-possible code in a client

Lastly, Flag indicates the value for a response type as shown in Table 5. Success in Table 5 means a request success as well as messages that record the processed results in Content and send them out. Error, in the occurrence of error found during the execution, inserts a substitute code in the client and

makes requests to the server. It also helps to get the server to change the code and perform an operation. Redirect is a message that notifies the client of the fact that a request is transferred from the in-process server to another server. Then, the client is reconnected to the modified server address and becomes communicable after receiving redirect messages. Post is used when the server makes a request to the client for code execution or when renewing the client's user interface (UI) or current situations.

3.3 Cloud Computing Service

The system in this research is capable of implementing requests simultaneously in several servers using WebSocket. HTML5 WebSocket performs a 3-handshake to connect to the server. Connection is then confirmed only for a client with a right to perform, and therefore, a request for an anonymous client can be withdrawn in advance. Fig. 6 depicts the signal flow of the cloud computing system. For an initial operation, 3-handshake is implemented so it can connect to the server using WebSocket. The service can be easily used as it provides possible real-time two-way communication. The next step involves the client using a WV message to request a code to execute to the server. The server that receives the response checks if the client is authorized and processes it immediately. Then, it transmits the success response message to the client. The Event Trigger upon the receipt of the success message calls a response handling function assigned by the client. After checking the success message, the client carries out other work until the next Event Trigger occurs.

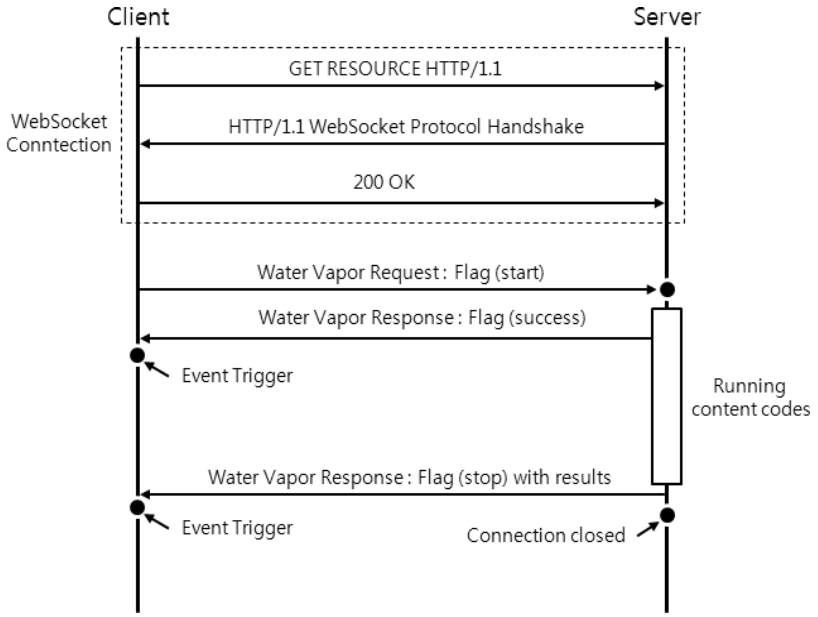


Fig. 6. The proposed system's signal flow graph.

After that, the server sends the final result by the response message, transmits the stop message to the client as a signal that the execution has ended, and immediately shuts down the connection to receive a request from the client. The client with the final notified results performs a task based on the results. While this helps obtain promptly the processed results of the complicated operations, in which the client should perform or the process requiring a lot of time, a client occasionally makes additional

requests for the work situation of the server or specific details. For example, when needing to know about how much the calculation has been finished so far or what is the intermediate result in operating 100! (*factorial*), one can request to the server. At this time, the question and answer can be done with the server using the request of the WV message for post.

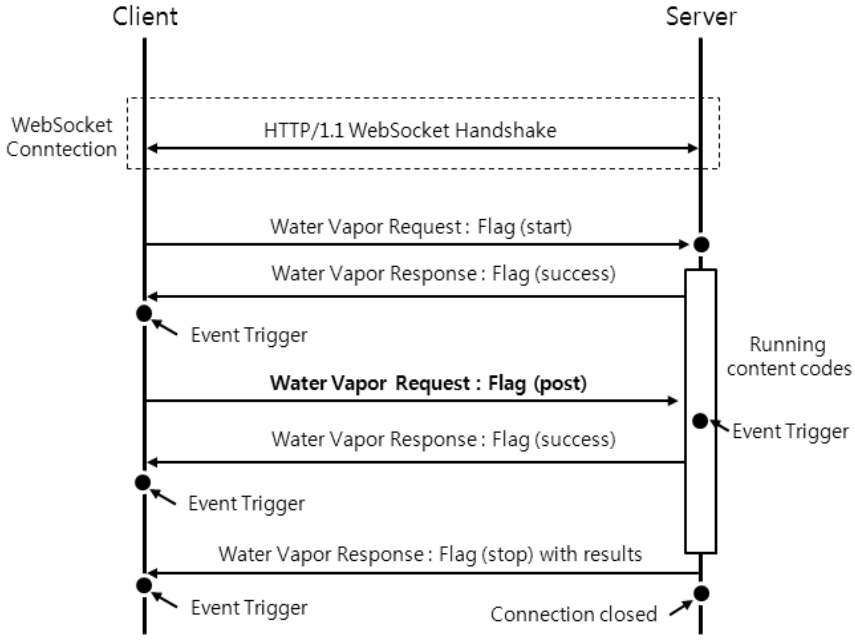


Fig. 7. Signal flow graph for processing a specific request from an operating server.

Fig. 7 displays this signal flowchart. The client in Fig. 7 can request the values for specific variables or user-defined data to the server being currently executed with the use of the post in the WV message. The Event Trigger of the server that received the request handles the requested signal processing and stores the values in the content in the success response message prior to transmitting. Then, the question and answer process is completed by calling the user-defined function by the Event Trigger of the client. In case the execution error occurs due to the wrong passage of the user code or the malfunction of the server system when executing in the server, the server immediately closes the handling process, transmits the error message and cuts off the connection. Fig. 8 shows the signal processing flowchart in the occurrence of such an error.

According to Fig. 8, upon the occurrence of an error in the server, it sends the WV error message to the client, exit the process being executed and immediately cut off the connection. However, when there is an error in the performance code transmitted from the client and exceptional handling such as the try-catch statement, it is not terminated but waits for a replacement code. Fig. 9 depicts a flowchart of signals requesting replacement codes in the error occurrence. The figure shows a process in which the client detects the error created during execution in the server, inserts the replacement code, and transmits WV POST messages to the server. The waiting server inserts the replaced request code from the client before continuing with the next process. When an error takes place again due to the replaced code, instantly close the process, send the error message and cut off the connection.

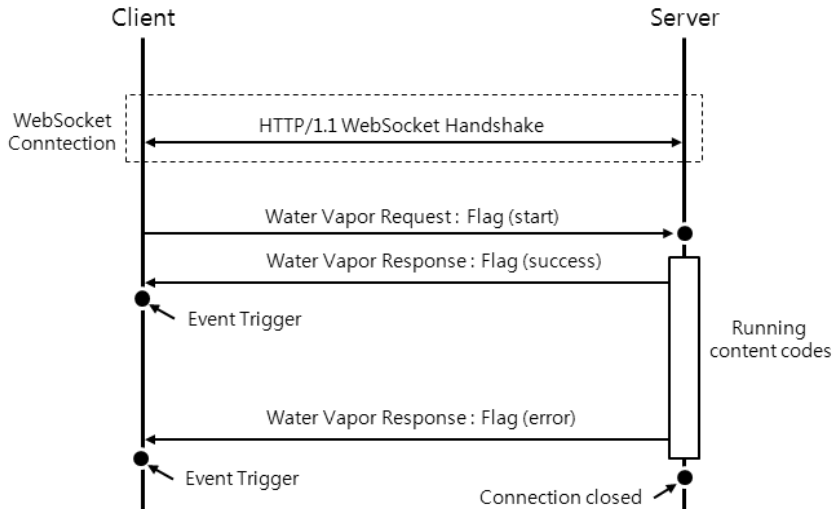


Fig. 8. Signal flow graph of an error case for an operating server.

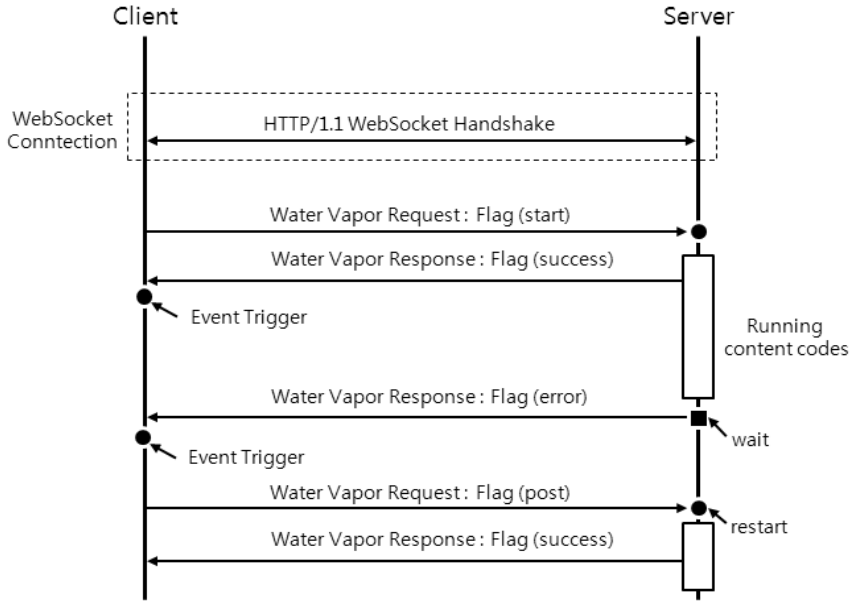


Fig. 9. Signal flow graph for an exceptional request processing of an error case for the operating server.

Unless the error comes up, the success message is delivered to the client. When the client makes requests to a number of servers for processing, the client can get the returned results during the process and the server request is instantly delivered to another server. Fig. 10 shows the signal flowchart of the system that implements the 1:N request. In Fig. 10, the request is made to server A and subsequent results are delivered to the client. Then, server A closes a connection and the client attempts to connect the WebSocket to the address of the following server in Hosts. Upon completion of the connection to server B, a request message is sent along with the code to implement next and result data received from

server A. Server B, which receives the request, ultimately delivers the results handled afterward to the client. The proposed cloud framework system is composed of a client-server model, and therefore, unlike JIT and Web Worker which run in local environments, the system performance somewhat depends on the amount of transferring data and network conditions between the client and the server. The system has a function that is able to expect the execution time dynamically with respect to current network conditions. Fig. 11 shows the process of pre-evaluation of running time.

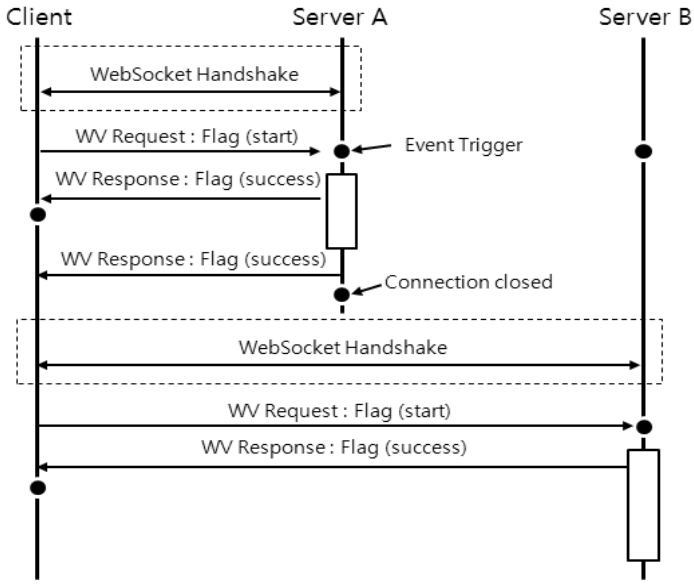


Fig. 10. Signal flow graph for a request and 1:N server connection.

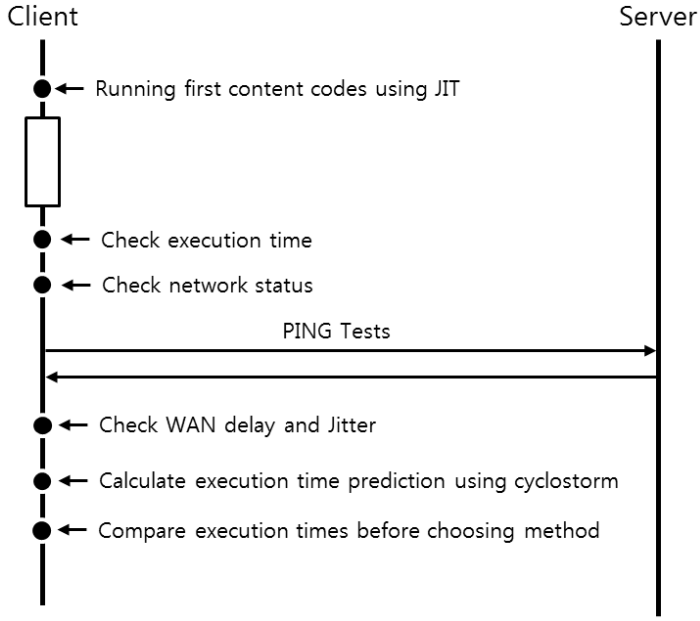


Fig. 11. Signal flow graph of pre-evaluation of the execution time.

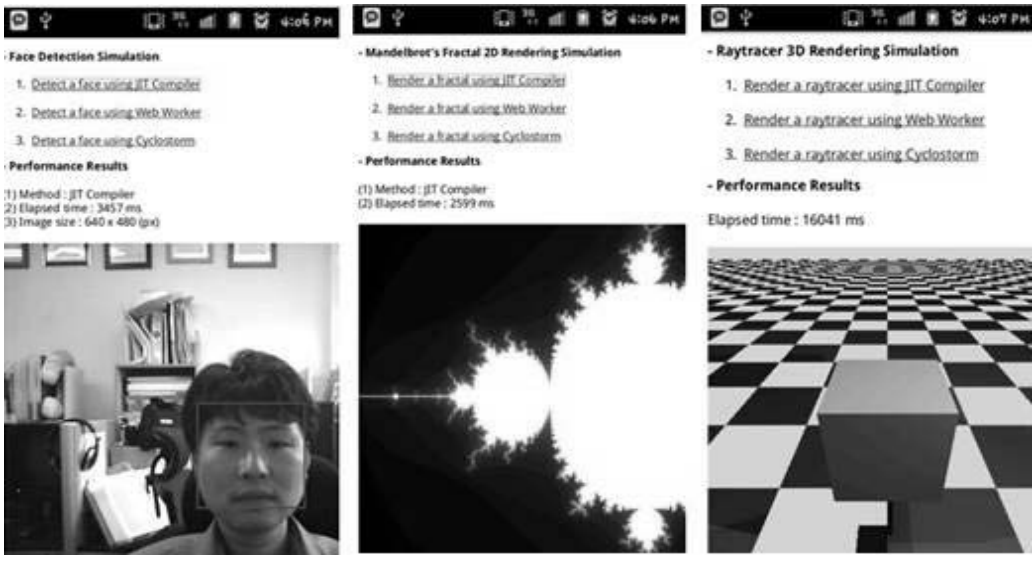
In order to decide whether to send a work to the cloud or not, the system measures the execution time by processing a function or a code of JavaScripts from the GSMA applications using the JIT. At this point, the JIT method is used as a standard since there is only '±2%' average functional difference between the JIT and Web Worker. Next, the data transmitting and receiving delay due to WAN delay and jitter is measured through a ping test between the client and the server. One advantage of the system studied in this research is that it can promptly process in the server the JavaScript code requiring too much time to be handled promptly under limited mobile client environments. JavaScript, in general, is a scripting language used in the client side for a simple task. The applications developed according to the GSMA standards, however, should be able to execute a variety of functions of native applications promptly and accurately. JavaScript working in the browser is largely dependent on hardware performance of a terminal in terms of its processing speed. As the mobile terminal with high-speed processing capability like smartphones increasingly improves its processing power close to the level of general desktop PC, this problem is expected to be resolved. But, at the present time when low-end specification terminals exist at all times and a software structure is becoming more and more complicated, the proposed system, Cyclostorm, is capable of providing all terminals with processing power with little difference from each other. A mobile terminal connected to the internet utilizes powerful hardware resources in the server side, which, therefore, can improve processing speed and reduce the processing burden of the client. The cloud computing services are expected to be usefully applied in that the present modern times allow most of mobile terminals to use wireless network such as 3G, 4G and Wi-Fi.

4. Simulation and Performance Assessment

4.1 Simulation System

HTML5-based simulation environment was built to assess the performances of the proposed system. The free and open-source web browser can be used for performance assessment in the client, for it supports Web Worker among HTML5. In addition, a simulation web application is built for the system performance assessment, using Face Detection, 2D Fractal rendering and 3D Raytracer rendering, which require a relatively large number of operations. Fig. 12 shows the specifications and basic information about the web applications used for the system's performance assessment. Such applications as Face Detection, 2D Fractal rendering, and 3D Raytracer rendering shown in Fig. 12 are all composed of JavaScript and use HTML5 canvas to process images. The applications require so many operations that they take up approximately over 2 seconds even in the hardware-mounted high-end specification desktop browser.

The researched system uses a method that processes the results in the server to be utilized in the client. The server used for the system's simulation was designed to make web service possible. Hypertext preprocessor (PHP) modules for Apache 2.0 and WebSocket services are installed in the programs used for web services. The JavaScript engine installed in the server is a V8 and has modules needed to support the system's services implemented. For the clients for simulation use, the terminals commercialized at present were used because the processing speed of the JavaScript engine in the browser varies according to hardware performances in the mobile terminal. For the system's performance assessment, four different types of smartphone models, shown in Table 6, were selected.



(a) (b) (c)

Fig. 12. Web applications used for performance evaluation of the proposed cloud computing system. (a) Face Detection, (b) 2D Fractal rendering, and (c) 3D Raytracer rendering.

The mobile terminal presented in Table 6 performed the web applications shown in Fig. 12 and assessed their performances. To measure run-time, the modules for this were implemented in the web applications. JIT compiler, Web Worker and the system researched in this paper were used to run simulation 100 times each, and the average run-time was measured and its results were analyzed. The experiments have been done in Wi-Fi (local area networks [LAN]) and mobile 3G (metropolitan area networks [MAN]) environments. Under the Wi-Fi and 3G environments, the ping test results of 64 bytes data transmitting-receiving time were less than 2 ms and 25 ms, respectively.

Table 6. Mobile terminals and specifications used for performance evaluation of the system

Device	CPU	RAM
D1	1.2 GHz dual core	1 GB (DDR2)
D2	1.5 GHz dual core	1 GB (DDR2)
D3	1.0 GHz dual core	512 MB
D4	1.0 GHz dual core	1 GB (DDR2)

4.2 Performance Assessment and Result Analysis

First of all, to decide whether to send a work to the cloud or not, the system measures a running time by processing a function or a code of JavaScripts from the GSMA applications using the JIT. Eq. (1) shows the way of computation of expected processing time, $T_{estimate}$. From the previous evaluation of the classification of works, if the estimated running time, $T_{estimate}$, of the proposed system is shorter than other two methods (JIT and Web Worker), then the Cyclostorm is used; otherwise, the JIT or Web Worker method is used. With the help of these processes, the GSMA applications are able to be performed more effectively.

$$T_{estimate}=[T_{JIT}\times(1-0.82)+D_{TR}] \quad (1)$$

The JIT processing time, T_{JIT}, in (1) is the duration of local resources' operation time of a terminal and D_{TR} is the network transmitting and receiving delay time obtained from the ping test. From the evaluation of processing time, $T_{estimate}$, the cloud framework system is 76.44% through 85.63% superior to other methods such as the JIT or Web Worker. It also showed 82.19% averaged efficiency improvements of reducing the processing time. Table 7 shows the average processing times from the experimental results.

Table 7. Comparison of the average processing time (10^{-3} second)

Device	JIT	WW	CY	Improved (%)	Ref. value
D1	4982.40	4750.70	1146.30	76.44	2433.2750
D2	7787.50	6684.40	1145.20	84.17	3617.9750
D3	7111.90	8867.20	1147.50	85.63	3994.7750
D4	6474.80	6613.40	1144.90	82.50	3272.0500
Avg.	6589.15	6728.92	1145.97	82.19	3329.5175

As it can be seen from the Table 7, the executing time of the proposed system can be estimated by multiplying 0.18 to the JIT processing time since there is 82.19% averaged improvements in processing speed. In addition, the network transmitting and receiving delay time, D_{TR}, is added from the ping test and then the total estimated running time, $T_{estimate}$, is obtained as shown in Eq. (1). Here we have reference values to decide whether to use the Cyclostorm. The reference value is set to 50% of the average expected operation time when using JIT or Web Worker method. Therefore, it only works when the expected operation time of the Cyclostorm can guarantee at least 50% improvement over the existing JIT or Web Worker method. This reference value is also shown in Fig. 7 as a basis for judging the operation of the Cyclostorm. It is recommended that the Cyclostorm is used if the average processing time of the system based on the log data is faster than that of local method. In addition, the average value of CPU and memory usage is measured and stored in the log data, and the current usage of each mobile device is compared to determine which method is better. On average, the Cyclostorm are CPU-intensive and use less memory, so using an average of 82% faster Cyclostorm is more efficient if they are heavily weighted on mobile device's processing time. Fig. 13 shows the measurement results of run-time of Face Detection (FD), 2D Fractal (2D) and 3D Raytracer (3D) applications implemented in JavaScript on Wi-Fi using the given four different client terminals (D1 through D4).

JIT compiler method generally measures execution time differently depending on hardware specification of a terminal. D1 from Table 6, which is relatively higher-end device than other terminals, showed the shortest run-time according to measurement in Fig. 13. Web Worker supported by HTML5 was generally handled faster than the JIT compiler, but measurement times were not constant due to the multitasking environment. JIT compiler and Web Worker methods revealed comparatively little difference in performances. The method in this paper (indicated as CY in the figures) utilizes high-performance hardware present in the server side without being affected by the multitasking environment, which, therefore, brought a prompt processing in all terminals. According to Fig. 13, the run-time for rendering 2D Fractal is not as long as the Face Detection application program, but performance assessment results from measurement showed that D1 with the outstanding hardware

performance was the fastest in run-time. As the 3D case had a larger amount of operation processed than Face Detection and 2D Fractal, the processing time was longer in all terminals according to the measurement. When compared to the JIT compiler and the Web Worker method processed in the local browser of the terminal, the proposed method was processed rapidly because it shows the results executed only in the server. Additionally, in 3D Raytracer rendering, it has less decimal point operations and data amount processed in the multi-thread than 2D Fractal. Thus, the Web Worker method drew results faster than the JIT compiler. Regarding the CPU share in Fig. 14, the JIT compiler and the Web Worker method were lower according to the measurement, when compared to the proposed method (CY).

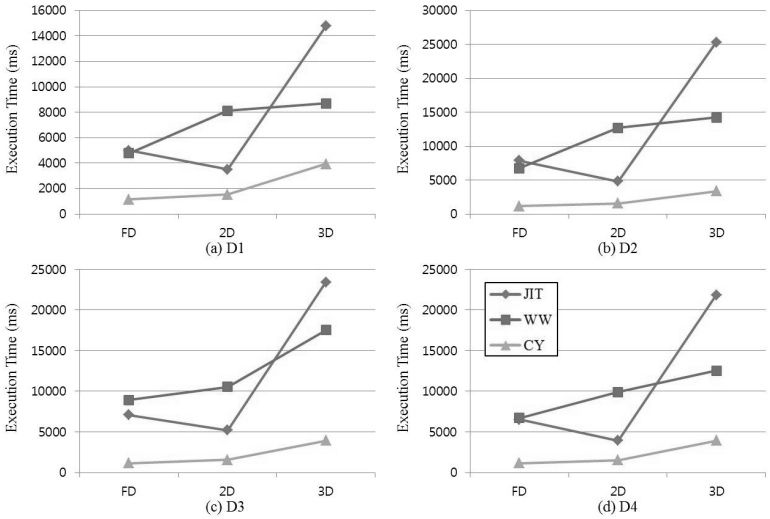


Fig. 13. Execution time at each smart devices for three different web applications (FD=Face Detection, 2D=2D Fractal, 3D=3D Raytracer). (a) D1, (b) D2, (c) D3, and (d) D4.

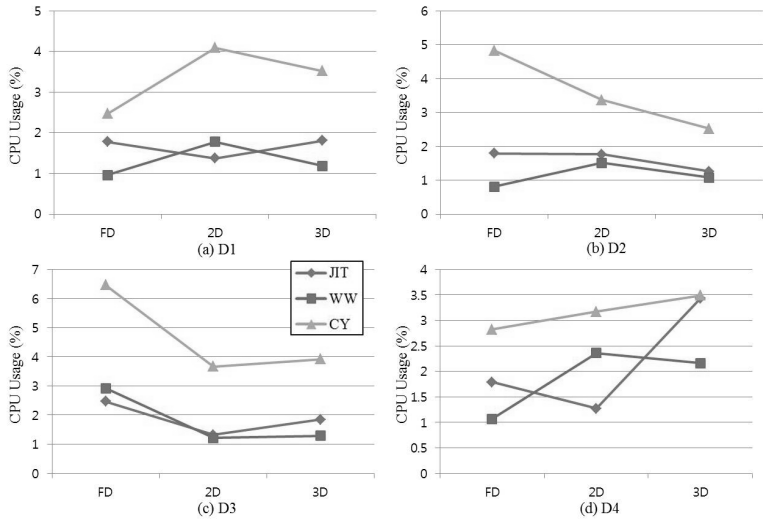


Fig. 14. CPU occupancy rate at each smart devices for three different web applications (FD=Face Detection, 2D=2D Fractal, 3D=3D Raytracer). (a) D1, (b) D2, (c) D3, and (d) D4.

This is because mobile network resources are used and the proposed framework is loaded while the system is connecting to the server to handle the web program and receiving the corresponding results. However, this CPU usage was 4% to 7%, which was comparable with the existing methods. In addition, since it maintains a very low usage when compared to the overall CPU usage, it does not affect other functions of a terminal, either. Rendering a 2D Fractal applies operations below decimal point in many cases, which can impose many hardware-related burdens in mobile terminals.

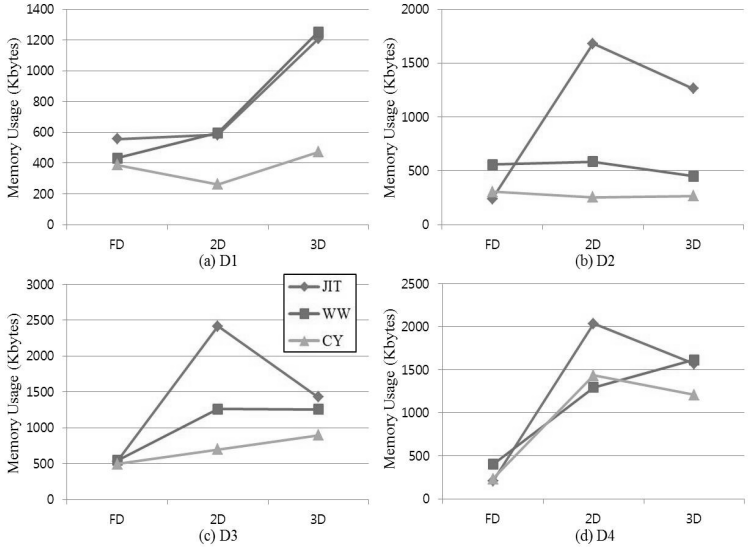


Fig. 15. Memory usage at each smart devices for three different web applications (FD=Face Detection, 2D=2D Fractal, 3D=3D Raytracer). (a) D1, (b) D2, (c) D3, and (d) D4.

This is why the JIT compiler running on the main terminal was handled faster than Web Worker, which creates multi-thread through a temporary pause of the operations employed in UI outputs as well as the allocation of many CPU resources only to the decimal point operation. Measurement results shown in Fig. 15 had the least amount of memory used by the CY. The devices' mobile platform uses Java, so the Garbage Collector is responsible for memory management, whereas the CY utilizes memories in the server side when executed. Thus, it seems that memory usage was generally lower than the other methods according to measurement. The memory used at time of execution was also used in the server side. Hence, the CY generally showed more excellent results than the other methods. Fig. 15 includes much of the basic amount of memory for 3D rendering needs presented a comparatively higher figure than other performance assessment measurements. Fig. 16 displays measurement and comparison results of processing times in the system, 3G and Wi-Fi environment using three different client terminals, with the exception of D4 (tablet) in Table 6 showing no support for 3G communication.

The 3G network environment is relatively slower in speed than Wi-Fi in terms of transmission, taking up longer processing time than Wi-Fi. However, measurement results showed a slight difference of approximately 10 to 400 ms between the two methods. It is assumed that the reasonable 3G network provides services at a speed level similar to Wi-Fi connected to a wired network. According to performance assessments of Face Detection, 2D Fractal, and 3D Raytracer rendering, the processing

speed of the system was at 3 to 6 times faster on the Wi-Fi network than the JIT compiler and the Web Worker method. Additionally, the use of proposed method enables a similar processing time to be measured, regardless of hardware in mobile environments, and complicated JavaScript-embodied operation or rendering enables more effective processing than the existing method and is capable of providing services with performance in the identical level. The system in this paper yielded comparative advantages over other the methods as it utilizes the memory used at time of execution in terms of memory usage.

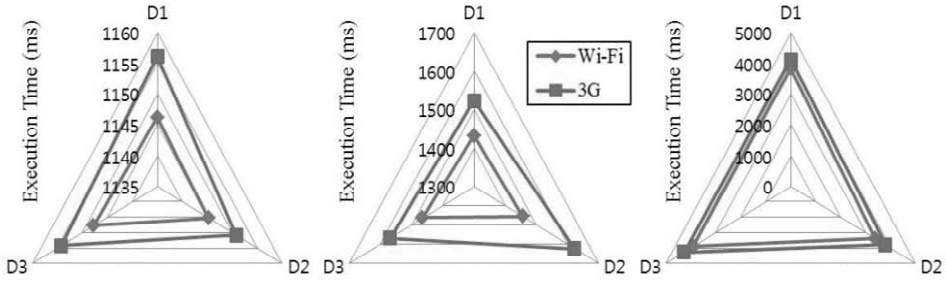


Fig. 16. Execution time at each smart devices for three different web applications in Wi-Fi and 3G environments. (a) Face Detection, (b) 2D Fractal, and (c) 3D Raytracer.

5. Conclusions

With a rise in the global distribution of GSMA based mobile web application software, the processing capability of the application embodied with JavaScript and HTML5 is an increasingly relevant and important issue. If it is a structure with simple processing functions, there is no problem in the currently commercialized browser. However, the continuous growth of the processing capacity of JavaScript for communication with the user also increases the browser burden. A commercialized mobile browser is limited in time and capacity in JavaScript processing. There are limitations in browser processing and, especially, 3D application that demands many operations to be handled, which does not guarantee that the processing speed will be as fast as the native application. As an alternative, HTML5 provides the Web Worker for multi-thread implementations not supported by the existing JavaScript. Web Worker provides a mechanism that processes a certain part of what a single thread processes through a separate thread. But, it may not guarantee the processing power of the native application and is insufficient in improving the fundamental speed of processing. Moreover, when the mobile hardware has low specifications, Web Worker alone is not enough to reduce the burden of hardware. The system in this paper is a service that overcomes the restrictions of limited sources of a client by transferring the JavaScript processing on the mobile to a cloud-based server. This service also provides a high-performance handling process to guarantee as much performance as the native application holds. In a performance assessment test, the proposed system was approximately 6 times faster in performance than JavaScript processing on the existing mobile browser, displaying about 4 to 6 times faster performance than Web Worker. CPU usage was in the range of 4% to 7%, which was lower than the overall usage, with little effect on the terminal performance. In addition, a low amount of usage

was generally found because it utilizes the memory used at time of execution kept in the server side. Performance comparison and assessment results indicated that the proposed system is capable of overcoming the limitations the browser has and considerably improving existing web application functions. It also showed to be offering services with no major difference even though 3G is relatively slower in speed than Wi-Fi. Continual future research studies are planned on effective cloud computing that can be applied to the proposed system also in 4G and future mobile environments.

Acknowledgement

The present research was conducted by the Research fund of the Small and Medium Business Administration of Republic of Korea in 2017. (No. S2444403)

References

[1] J. Meyer, *HTML5 and JavaScript Projects*. New York, NY: Apress, 2011.

[2] A. Taivalsaari and K. Systa, "Cloudberry: an HTML5 cloud phone platform for mobile devices," *IEEE Software*, vol. 29, no. 4, pp. 40-45, 2012.

[3] A. MacCaw, *JavaScript Web Applications*. Sebastopol: O'Reilly Media, 2011.

[4] J. K. Martinsen, H. Grahn, and A. Isberg, "Using speculation to enhance JavaScript performance in web applications," *IEEE Internet Computing*, vol. 17, no. 2, pp. 10-19, 2013.

[5] B. S. Yang, J. Lee, S. Lee, S. Park, Y. C. Chung, S. Kim, K. Ebcioglu, E. Altman, and S. M. Moon, "Efficient register mapping and allocation in LaTTe, an open-source Java just-in-time compiler," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 1, pp. 57-69, 2007.

[6] C. Rohlf and Y. Ivnitskiy, "The security challenges of client-side just-in-time engines," *IEEE Security & Privacy*, vol. 10, no. 2, pp. 84-86, 2012.

[7] V. Pimentel and B. G. Nickerson, "Communicating and displaying real-time data with WebSocket," *IEEE Internet Computing*, vol. 16, no. 4, pp. 45-53, 2012.

[8] I. Green, *Web Workers: Multithreaded Programs in JavaScript*. Sebastopol: O'Reilly Media, 2012.

[9] Y. Watanabe, S. Okamoto, M. Kohana, M. Kamada, and T. Yonekura, "A parallelization of interactive animation software with web workers," in *Proceedings of the 16th IEEE International Conference on Network-Based Information Systems*, Gwangju, Korea, 2013, pp. 448-452.

[10] P. Lubbers, B. Albers, and F. Salim, *Pro HTML5 Programming*, 2nd ed. New York, NY: Apress, 2011.

[11] L. Ullman, *Modern JavaScript: Develop and Design*. San Francisco, CA: Peachpit Press, 2012.

[12] D. Tiwari and D. Solihin, "Architectural characterization and similarity analysis of sunspider and Google's V8 JavaScript benchmarks," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, New Brunswick, NJ, 2012, pp. 221-232.

[13] Wikipedia, "JavaScript," 2011 [Online]. Available: https://en.wikipedia.org/wiki/JavaScript.

[14] R. Radhakrishnan, N. Vijaykrishnan, and L. K. John, "Java runtime systems: characterization and architectural implications," *IEEE Transactions on Computers*, vol. 50, no. 2, pp. 131-146, 2001.

[15] C. Vivaracho-Pascual and J. Pascual-Gaspar, "On the use of mobile phones and biometrics for accessing restricted web services," *IEEE Transactions on Reviews*, vol. 42, no. 2, pp. 213-222, 2012.

[16] C. Severance, "Discovering JavaScript object notation," *Computer*, vol. 45, no. 4, pp. 6-8, 2012.

[17] S. S. Sriparasa, *JavaScript and JSON Essentials*. Birmingham, UK: Packt Publishing, 2013.
[18] A. Gal, C. W. Probst, and M. Franz, "HotpathVM: an effective JIT compiler for resource-constrained devices," in *Proceedings of the 2nd VEE International Conference on Virtual Execution Environments*, Ottawa, Canada, 2006, pp. 144-153.
[19] B. Gao, L. He, and S. A. Jarvis, "Offload decision models and the price of anarchy in mobile cloud application ecosystems," *IEEE Access*, vol. 3, pp. 3125-3137, 2016.
[20] A. Gheith, R. Rajamony, P. Bohrer, K. Agarwal, M. Kistler, B. L. White Eagle, C. A. Hambridge, J. B. Carter, and T. Kaplinger, "IBM Bluemix mobile cloud services," *IBM Journal of Research and Development*, vol. 60, no. 2-3, pp. 1-12, 2016.
[21] F. Y. Jiang and H. C. Duan, "Application research of WebSocket technology on web tree component," in *Proceedings of the IEEE Symposium on Information Technology in Medicine and Education*, Hakodate, Japan, 2012, pp. 889-892.
[22] N. Serrano, J. Hernantes, and G. Gallardo, "Mobile web apps," *IEEE Software*, vol. 30, no. 5, pp. 22-27, 2013.
[23] S. Kurumatani, M. Toyama, and E. Y. Chen, "Executing client-side web workers in the cloud," in *Proceedings of the 9th IEEE Asia-Pacific Symposium on Information and Telecommunication Technologies*, Santiago & Valparaiso, Chile, 2012, pp. 1-6.
[24] X. Ma, Y. Zhao, L. Zhang, H. Wang, and L. Peng, "When mobile terminals meet the cloud: computation offloading as the bridge," *IEEE Network*, vol. 27, no. 5, pp. 28-33, 2013.
[25] Y. Wu, Z. Zhang, C. Wu, Z. Li, and F. C. M. Lau, "CloudMoV: cloud-based mobile social TV," *IEEE Transactions on Multimedia*, vol. 15, no. 4, pp. 821-832, 2013.
[26] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in mobile cloud computing: taxonomy and open challenges," *IEEE Communications Surveys &Tutorials*, vol. 16, no. 1, pp. 369-392, 2014.
[27] L. A. Tawalbeh, R. Mehmood, E. Benkhlifa, and H. Song, "Mobile cloud computing model and big data analysis for healthcare applications," *IEEE Access*, vol. 4, pp. 6171-6180, 2016.
[28] I. Bojanova, J. Zhang, and J. Voas, "Cloud computing," *IT Professional*, vol. 15, no. 2, pp. 12-14, 2013.
[29] B. Frankston, "HTML5," *IEEE Consumer Electronics Magazine*, vol. 3, no. 2, pp. 62-67, 2014.

Daewon Kim http://orcid.org/0000-0001-6964-9535

He received a M.S. (1996) from the University of Southern California, Los Angeles, CA, USA, and a Ph.D. (2002) in Electrical and Computer Engineering from Iowa State University, Ames, IA, USA. He worked as a senior researcher at Samsung Electronics Co. Ltd., Suwon, Korea (2002–2004). He is currently a professor in Department of Applied Computer Engineering at Dankook University, Korea. His research interests include signal processing, mobile applications, and nondestructive evaluation.