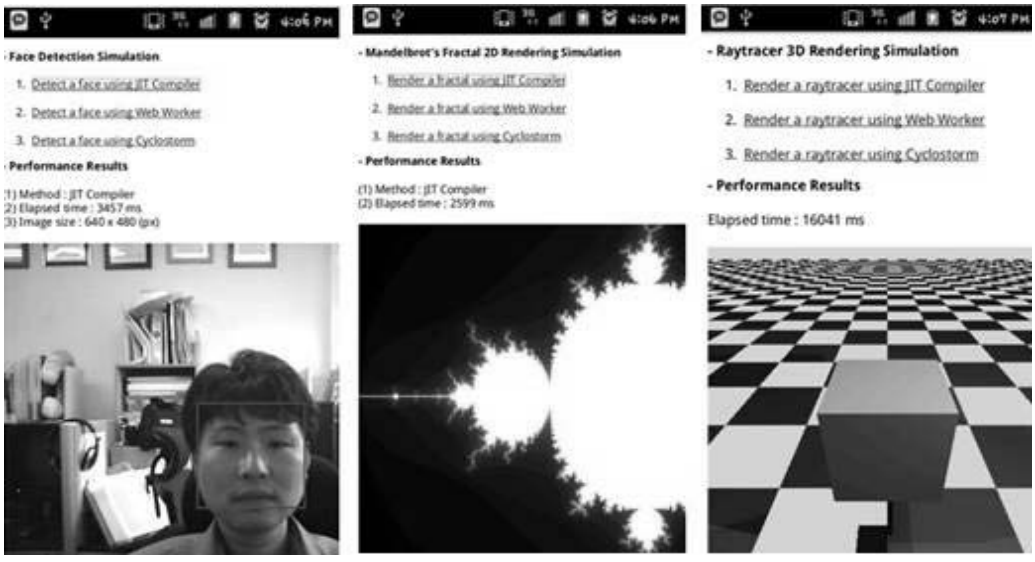

(a) (b) (c)

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).

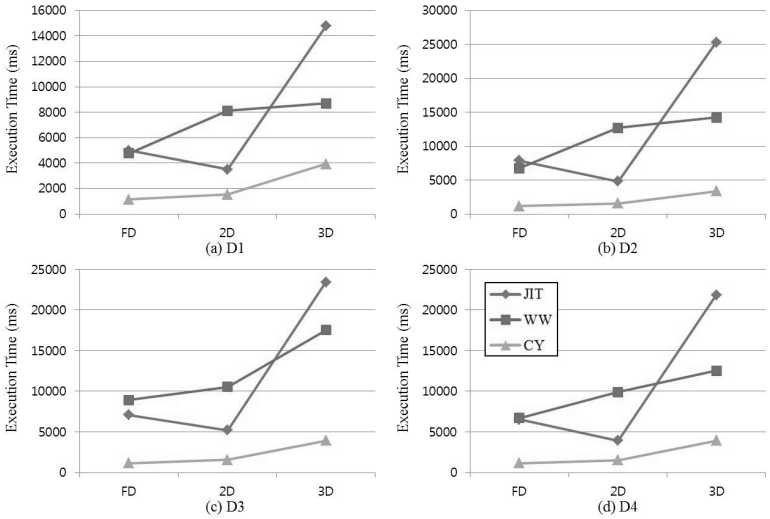

Fig. 13. Execution time at each smart devices for three different web applications (FD=Face Detection, 2D=2D Fractal, 3D=3D Raytracer). (a) D1, (b) D2, (c) D3, and (d) D4.

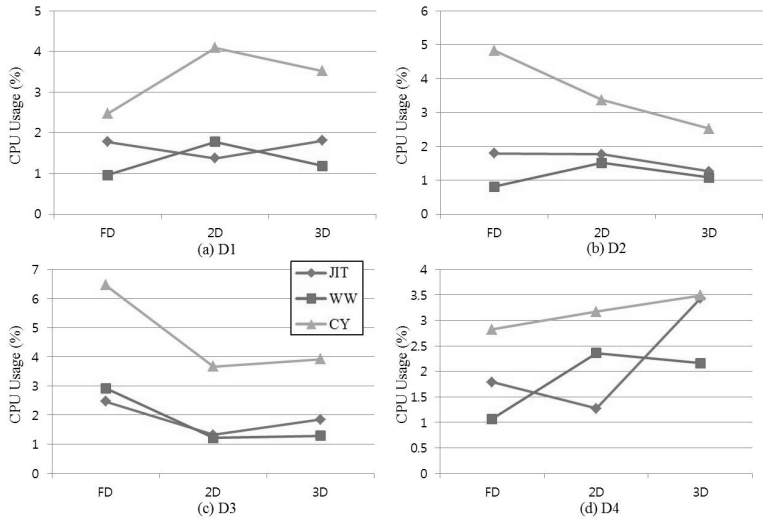

Fig. 14. CPU occupancy rate at each smart devices for three different web applications (FD=Face Detection, 2D=2D Fractal, 3D=3D Raytracer). (a) D1, (b) D2, (c) D3, and (d) D4.

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.

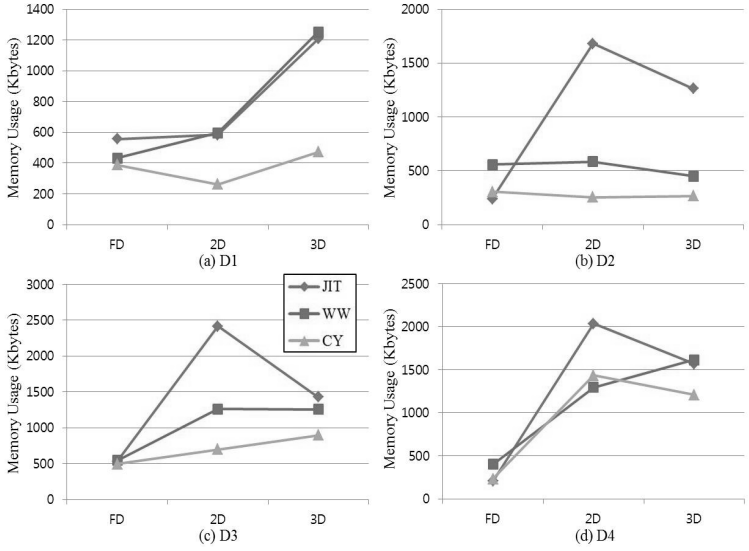

Fig. 15. Memory usage at each smart devices for three different web applications (FD=Face Detection, 2D=2D Fractal, 3D=3D Raytracer). (a) D1, (b) D2, (c) D3, and (d) D4.

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.

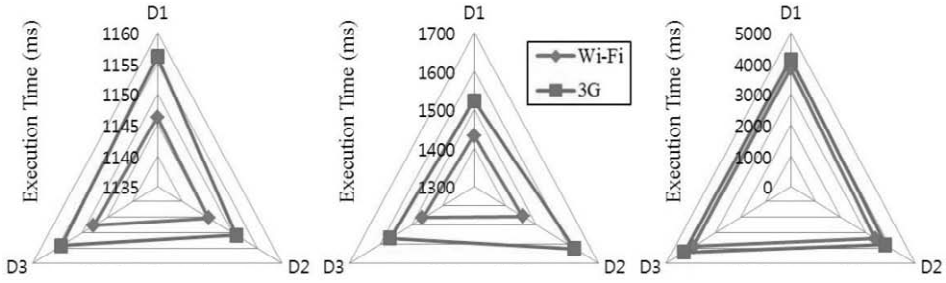

Fig. 16. Execution time at each smart devices for three different web applications in Wi-Fi and 3G environments. (a) Face Detection, (b) 2D Fractal, and (c) 3D Raytracer.

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.