J Inf Process Syst, Vol.17, No.1, pp.191~202, February 2021
https://doi.org/10.3745/JIPS.01.0064

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

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

On-Demand Remote Software Code Execution Unit Using On-Chip Flash Memory Cloudification for IoT Environment Acceleration

Dongkyu Lee*, Moon Gi Seok**, and Daejin Park*,***

Abstract

In an Internet of Things (IoT)-configured system, each device executes on-chip software. Recent IoT devices require fast execution time of complex services, such as analyzing a large amount of data, while maintaining low-power computation. As service complexity increases, the service requires high-performance computing and more space for embedded space. However, the low performance of IoT edge devices and their small memory size can hinder the complex and diverse operations of IoT services. In this paper, we propose a remote on-demand software code execution unit using the cloudification of on-chip code memory to accelerate the program execution of an IoT edge device with a low-performance processor. We propose a simulation approach to distribute remote code executed on the server side and on the edge side according to the program's computational and communicational needs. Our on-demand remote code execution unit simulation platform, which includes an instruction set simulator based on 16-bit ARM Thumb instruction set architecture, successfully emulates the architectural behavior of on-chip flash memory, enabling embedded devices to accelerate and execute software using remote execution code in the IoT environment.

Keywords

Edge-Side Acceleration, Memory Cloudification, On-Demand Remote Code Execution

1. Introduction

Recently, the Internet of Things (IoT) has reached the stage in which an artificial intelligence (AI) system can be embedded away from simple data processing at the edge [1–4]. This embedded AI system provides services tailored to the user's characteristics at the embedded edges [5,6]. The edge used in an embedded AI system must embed the full code of the service and requires a processor to execute the service. In other words, the edge needs much power and a large hardware size to support a large memory size and high-end processors and accelerators [7–11]. However, embedded IoT devices have a small hardware size and power-consumption limitations for portability [12,13]. The embedded system has several disadvantages. All services must run on one edge, so they must be designed by the same hardware architecture and the same software structure. Each service provider needs to consider the interactions with other services when creating a service. Hence, previous systems are not scalable because all the connection code between

※ This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.
Manuscript received March 19, 2020; first revision June 22, 2020; accepted July 3, 2020.
Corresponding Author: Daejin Park (boltanut@knu.ac.kr)
* School of Electronic and Electrical Engineering, Kyungpook National University, Korea (ehdrbxp@gmail.com, boltanut@knu.ac.kr)
** School of Computer Science and Engineering, Nanyang Technological University, Singapore (moongi.seok@ntu.edu.sg)
*** School of Electronics Engineering, Kyungpook National University, Korea

Copyright© 2021 KIPS

services must be implemented at the base code, which is be embedded at the edge [14–16]. This is not secure because the server requires full code of the service to integrate the services into the base code [17–22]. Distributed computing is studied to solve the problems that occur because all code is stored in the edge device [23]. It diversifies the services that an edge can provide because it shares the software of the server and the edge. Each software is executed in the owning device, so the security is guaranteed. However, it needs many data transmissions for executing programs in distributed code, and complex algorithms to control the heterogeneous systems. We propose a remote code execution unit that provides flexible services with a small hardware size. This proposed unit provides flexible services by selecting codes stored in the server in real time according to the internal configuration. It can also reduce execution time by using various resources on the server with small size edge hardware.

This paper proposes a method that distributes the service code with the cloud server. The method executes the parts of the edge's service code in remote servers through the cloudification of on-chip flash memory. On-chip flash memory cloudification aims at the hybrid computation of on-chip remote execution and on-cloud remote execution [24–26]. On-chip remote execution means that the service code on the server, which is requested by the edge, is executed at the edge's processor. On-cloud execution means that the service code is executed at the server, and the results are sent to the edge [27,28]. The on-chip execution code does not require data communication time because the embedded device can immediately process the collected data, but the program execution time is long due to the embedded device's low-performance. The on-cloud execution code, on the other hand, allows the server's high-performance processors and accelerators to be used, which speeds up the program execution but requires data-transmission time [29]. To reduce the program execution time, the ratio of the on-chip execution code to the on-cloud execution code must be efficiently distributed [30]. The distribution ratio is derived using the proposed simulation platform.

This paper is organized as follows: Section 2 describes the cloudification of the on-chip flash memory simulation structure for acceleration in an IoT environment. In Section 3, we analyze the simulation results in the proposed simulation structure. Section 4 concludes this paper.

2. Proposed Architecture

In this paper, we propose a method of solving code memory shortage and slow execution speed due to the small size of an embedded device using remote execution code. Fig. 1 shows the execution flow of the software embedded at the edge. As shown in Fig. 1, the main code selects multiple services according to the conditions and accesses them through different interfaces for each service. The complexity and size of the embedded software increase as the services are added, due to the overlapping structure of the many branch statements. Each service requires the interface code to access the service and the stub code for interaction between services.

Fig. 2 shows the process of adding or modifying a service in the main code. If the service company develop the service on a different hardware architecture, then it must be cross-compiled with the same architecture as the edge device on which the program is executed. To make one program using different interface for each company, the stub code must be directly designed in the main code. After using the stub code to implement the service code into the main code, all the main code must be built. If one service is modified, then the embedded code has low scalability because the entire service must be re-built after

the service is modified. The method of recompiling or designing the stub code creates a security problem because the service companies must send the entire source file to the company that designed the main code.

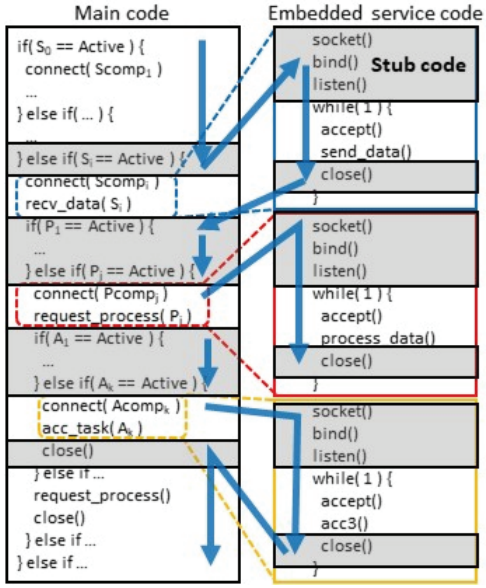


Fig. 1. The execution flow of the software embedded in the edge.

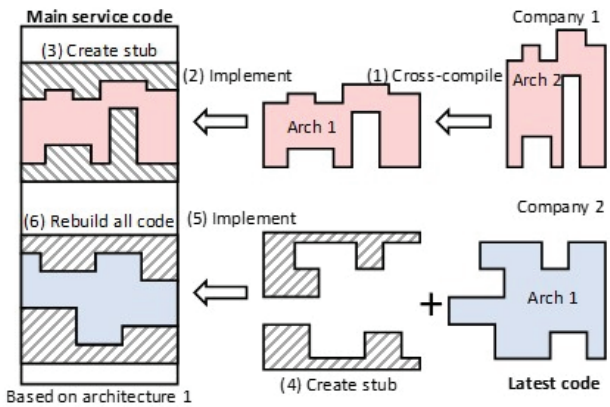


Fig. 2. The process of adding new services.

Fig. 3 shows the execution flow of the remote on-demand code used in this paper. In the remote execution code, the functions and connection information of each service's code and the main code are separated. Each service's code is designed with a different architecture and code structure for each company and creates a communication interface according to the stub virtual procedure interface (VPI). The stub VPI helps to access the same kind of services using the same interface when connecting to the service from the main code. The main code has a simple code structure without conditional statements for selecting a service through the stub VPI. The connection information for selecting a service is described separately to help the stub VPI allocate a desired service to a simplified interface. The service

requested by the main code is executed on the server, where the service is located, and only the results are delivered to the edge, so the service company does not need to match the same hardware architecture and code structure with those of the edge device. Moreover, the service company does not need to share the source code to design the stub of the main code, thus providing excellent security. The remote code execution is highly scalable. Even if a service is added or modified, it is not necessary to build the entire system.

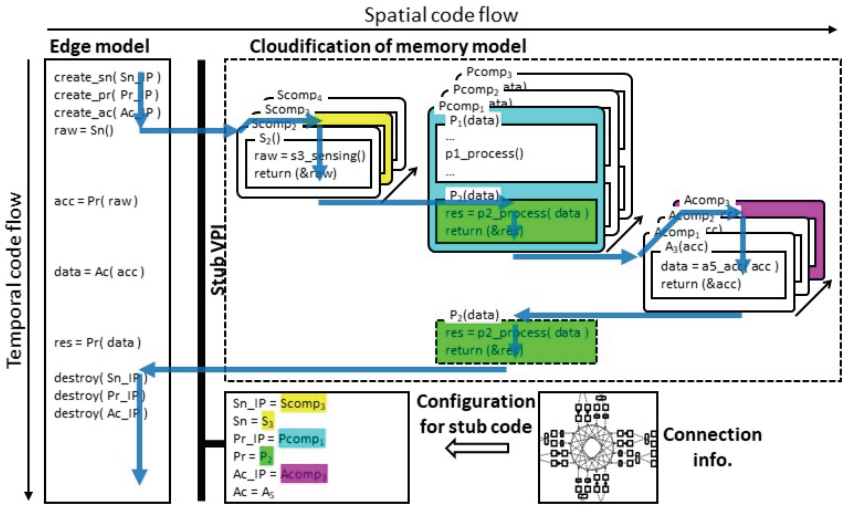


Fig. 3. The execution flow of the remote on-demand code execution.

Fig. 4 shows the interface structure of the base code and the service code when using remote execution code. Fig. 4(a) shows an IoT system in which the edge using the remote execution code and the server providing the service are connected in complex. Because the interface of the base code, which is executed on the edge, is different from the service, which is provided by the server, the developer must design the stub code between the interface of all services and the base code to create the integrated embedded software. Using the remote execution code, as shown in Fig. 4(b), the service developer creates an interface according to the remote execution code's VPI. The embedded edge creates a configuration file consisting of the service information and the connection information by selecting the services suitable for a given situation, as shown in Fig. 4(c). The interface, which is simplified by the stub VPI, can easily make a connection between the base code and service code with the configuration file.

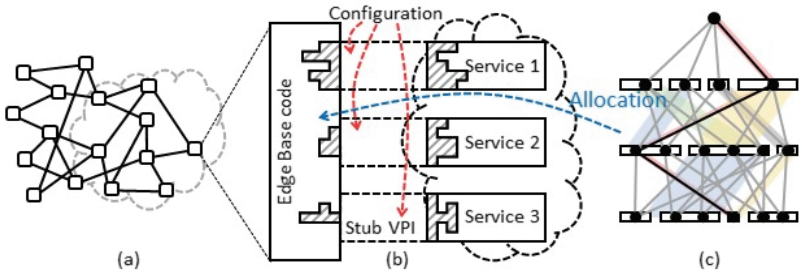


Fig. 4. The interface structure of the base code and the service code using remote execution code: (a) connected IoT, (b) software interfaces, and (c) calling connection.

Fig. 5(a) shows the concept of edge-side remote on-demand code execution (ROCE). The processor requires four steps to execute the instruction: instruction-fetch, instruction decode, execution, and write-back. The on-chip remote execution basically runs the code on its own processor. During the instruction-fetch stage, the ROCE client requests the code from the server. The server receives the request and transmits the code in the repository through the ROCE server. The arithmetic logic unit (ALU) of the edge executes code received through the ROCE client, and the ROCE client and the ROCE server prepare the next code to be executed by the ALU; thereby, the edge's on-chip flash memory is cloudified. Using the edge-side ROCE, the embedded device uses the server's code storage as virtual memory, which reduces the limitation of the program size, that the embedded device can execute on it.

The concept of server-side ROCE is shown in Fig. 5(b). In the instruction-fetch stage, the edge executes the server-side remote instructions. The ROCE client requests the service at the server in the same way as in edge-side ROCE. The server searches for requested code in the repository, executes it using its processor and accelerators, and transmits the results to the edge through the ROCE server. From the perspective of the embedded edge, the server-side remote execution code seems to do many operations in one instruction, like a complex instruction. With server-side ROCE, the embedded device uses the server as a hardware accelerator that processes a service. Because an operation that requires significant power is executed on a high-speed server, the embedded device utilizing server-side ROCE reduces both the execution time and the power consumption. Because the edge-side ROCE processes the service code at the edge, no data-transfer time is required, but the execution time is long due to the low-performance processor. In contrast, server-side ROCE involves processing the service code on the server using a high-performance processor, so the execution time is short, but it requires data-transmission time. Therefore, to reduce the execution time, the ratio of the edge-side remote code to the server-side remote code must be controlled depending on the program.

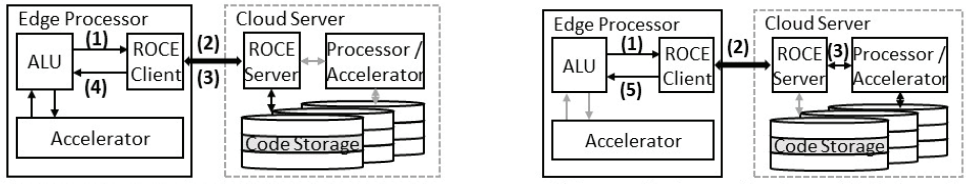


Fig. 5. The concept of the remote on-demand software: (a) edge-side remote on-demand code execution and (b) server-side remote on-demand code execution.

We designed a platform to efficiently partition the ratio of the edge-side remote code to the server-side remote code. Fig. 6 shows the platform used to find the ratio of the edge-side remote execution code to the server-side remote execution code. It consists of a connection layer, a code-generation layer, a simulation layer, and a reporting layer. The hardware-connection information of the IoT system to be simulated is implemented at the simulation layer. The connection layer receives the hardware-connection information from the simulation layer and adjusts the ratio of the edge-side remote code to the server-side remote code in order to make the program's virtual connection structure. The code-generation layer generates the simulation code using virtual connections created at the connection layer. The simulation layer simulates the code using hardware information. The platform iterates between modeling and simulation, consisting of the connection layer, code-generation layer, and simulation layer, while

controlling the ratio of the server-side remote code to the edge-side remote code. Then, the report layer shows the results of the repeated simulation. The user can find the optimal ratio of the code in the report layer.

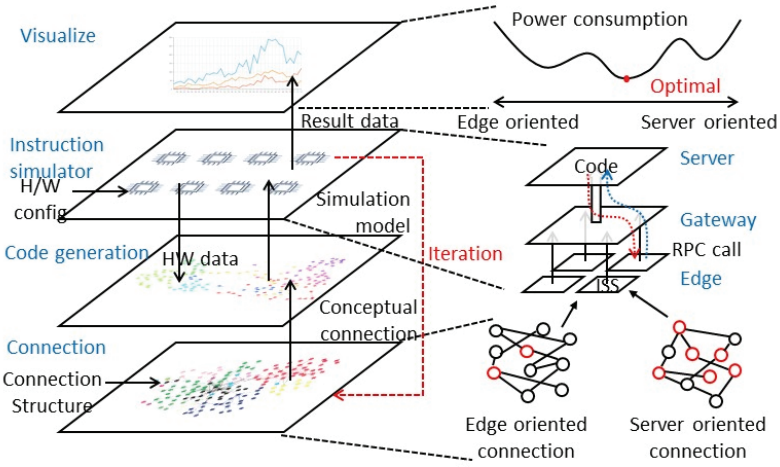


Fig. 6. The layer structure of the simulation platform with ROCE.

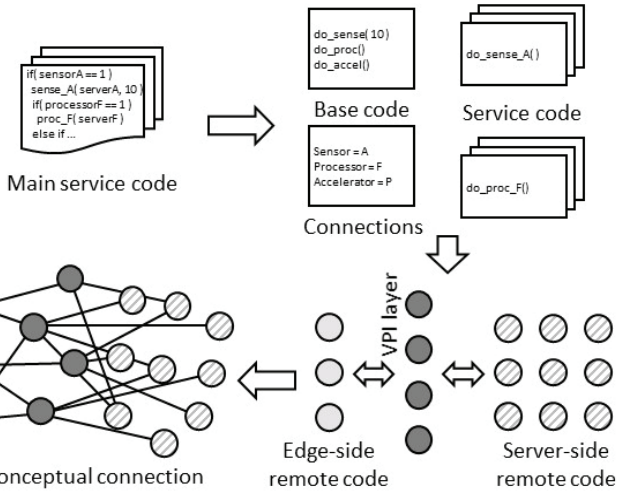


Fig. 7. The process by which the conceptual connections are made in the connection layer.

As shown in Fig. 7, the connection layer separates the entire main code into service code, base code, and connection information. The separated code is generated as the edge-side remote execution node, the server-side remote execution node, and the VPI layer node according to the ratio's parameter. The generated nodes are arranged and connected according to the hardware connection information given by the simulation layer.

Each service consists of static code and dynamic code. The static code does not change even if the connection changes, and it contains information about the program's operation. The dynamic code has the connection information, so it changes when the connection changes. In the code-generation layer, each node's static code and dynamic code are separated, and the simulation code is created by changing

the dynamic code according to the conceptual connection. In the simulation layer, the simulation code created in the generation layer is simulated through the real hardware or an emulator. The base code is embedded in the edge to execute the program. When the edge-side remote code is executed from the base code, the request is sent to the server through the gateway. The server transmits the requested remote code to the edge, and the edge executes the code through its own processor. When executing the server-side remote code from base code, the requested server uses its own processor and accelerator. Then, the result is passed to the edge.

We designed an instruction set simulator (ISS) based on the 16-bit ARM Thumb instruction set architecture (ISA) to emulate many edges and servers. Fig. 8 shows the concept of the designed ISS. The ISS successfully emulates the behavior of the simple ARM processor, which fetches and executes binary code. Because this paper's aim is to execute a program using remote execution code, the ISS basically receives and executes the code from the cloudified memory for program execution. The code fetched from the cloudified memory is temporarily stored in the random access memory (RAM). The stored program code is decoded and executed according to the program counter (PC) in the ISS. The analyzer measures the power consumption and execution time of the processor emulated by the ISS.

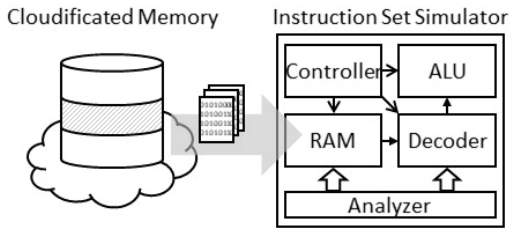


Fig. 8. The concept of the instruction set simulator based on ARM Thumb instruction set architecture.

Fig. 9 shows the remote execution code system when ISS emulates the edge. The edge requests the desired program code through the remote procedure call (RPC) caller. The gateway receives requests at each edge through the RPC callee, and the scheduler sequentially attaches the connectors to the server. The connector controls the communication link between the server and the edge. After the connector is linked, the server processes the requests that arrive at multiple edges through the task manager. Edge-side remote execution code is found in the storage and sent through the connector. In the case of the server-side remote execution code, the code in the storage is executed at the server's processor with the accelerators. Then, the results are sent to the edge through the connector.

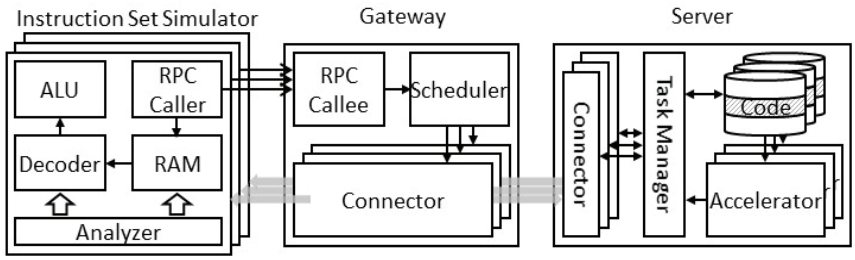


Fig. 9. The edge-server model for supporting multiple calls of remote execution code.

3. Experiments and Results

We implemented the proposed IoT system with 3 servers, 50 virtual edges, and 1 gateway and tested the effectiveness of the remote execution code. Three ODROID-XU4 devices were used to simulate the cloud server, and an ARTiGO A820 was used to simulate a gateway and the virtual edges. The ODROID-XU4 and ARTiGO A820 are based on the Ubuntu 16.04 operating system. Table 1 shows the CPU and memory specifications of the ODROID-XU4 and ARTiGO A820.

We implemented and simulated the proposed structure in the hardware environment shown in Fig. 10. The simulation used a program that executed n-dimensional square matrix data as a benchmark. The ARTiGO A820 device simulated the behavior of the gateway and the edges, and the ODROID-XU4 simulated the server's role. Using a program to multiply square matrices with the same numbers of rows and columns, we compared the execution time using the server-side execution code and the execution time using the edge-side execution code.

Table 1. Specifications of the ODROID-XU4 and ARTiGO A820

	ODROID-XU4	**ARTiGO A820**
CPU	ARM Cortex-A15 @ 2 GHz quad-core ARM Cortex-A7 quad-core	ARM Cortex-A9 @ 1 GHz
Memory	2 GB LPDDR3 RAM 14.9 GB/s	1 GB DDR3 SDRAM

Fig. 10. ODROID-XU4 and ARTiGO A820 for remote execution code simulation.

Fig. 11 shows the execution time according to the number of matrix rows. The horizontal axis represents the number of square matrix rows calculated by the program, and the vertical axis represents the program execution time. When the amount of calculated data is small (the number of rows is 10), executing the program on the edge-side is faster than doing so using server-side remote execution code. This is because it takes more time to transfer data from the edge to the server than the execution time that decreased due to computational acceleration. However, as the number of rows increases, more data need to be calculated, and the program needs to perform more, resulting in faster program execution times when executing the server-side remote execution code. The execution time for 50-row square matrix multiplication using the server-side remote execution code was 6.35 ms, which is about 50% less than the 12.56 ms when using the edge-side remote execution code.

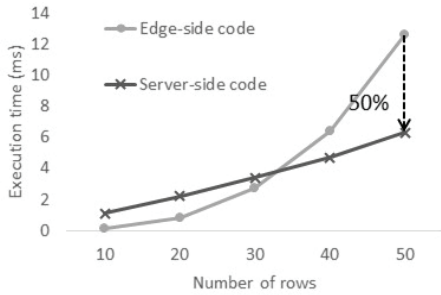


Fig. 11. Execution time according to the number of matrix rows.

Fig. 12 shows execution time results when using the edge-side versus server-side remote execution code, for which several edges are connected. We simulated the system that collects data from the edge's sensors and compares the data with other edges to process the data. When the program, which needed heavy computation, was executed by the edge-side remote execution code, it took 30.03 seconds because of the low performance of the processor at the edge. However, when the program was executed by the server-side remote execution code, the total execution time was reduced to 18.07 seconds because the computation times of the edge and the server decreased, even if the communication time increased. On the other hand, when the program with significant data exchange was executed by the edge-side remote execution code, the communication time was longer than the computation time at the edge. The computation time decreased when running the communication-intensive program using the server-side remote execution code. However, the computation time did not decrease much when viewed at full scale. Rather, the communication time increased, so the program's overall execution time also increased. As a result, the proposed platform accelerates code using server-side ROCE in a computation-intensive situation to reduce execution time. In a communication-intensive situation, the platform preprocesses the data using edge-side ROCE to reduce data transmission time, thereby reducing execution time. In future work, we will use the proposed platform to optimize the ratio of the edge-side remote code and the server-side remote code, and compare the execution time of the edge-side ROCE intensive program, the server-side ROCE intensive program, and the optimized ROCE in various IoT situations.

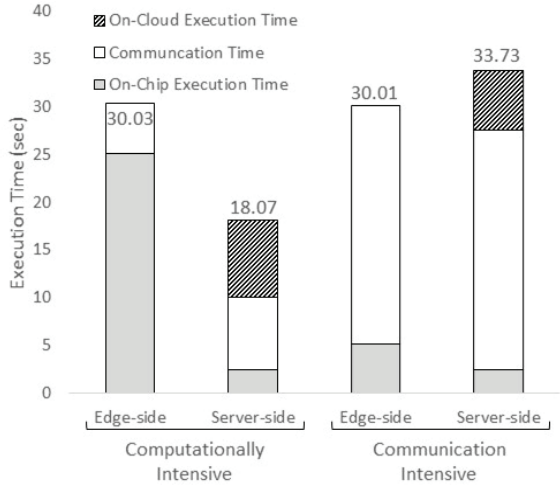


Fig. 12. Case study: the execution time of the edge-side and server-side remote execution code.

4. Conclusion

In this paper, we propose an IoT architecture that uses a server as an accelerator as well as on-demand remote execution code in embedded devices with small memory size and low power consumption. In the proposed structure, the embedded device represented by the edge executes the service requiring acceleration by using remote execution code. The remote execution code requested by the edge is sent to the server through the gateway. Each service company has a server with a repository that stores the service code and the processor to execute the code. The proposed system uses VPI to match the interface between the edge and the server, so that the remote code can select various codes with a simple configuration. The server requested to run the remote code executes the service using its processor and accelerators and transmits the results to the edge. When a program requiring a large amount of computation is executed in an embedded device using the proposed structure, the program execution time is reduced as when using a hardware accelerator. In a communication-intensive situation, the proposed system can reduce program execution time by increasing the ratio of edge-side ROCE to reduce the amount of data to be transmitted. The embedded devices can achieve short program execution time and low power consumption by using remote execution code, which acts as an accelerator within a limited hardware size. The cloudification of on-chip code memory helps to reduce the size of the memory in the embedded devices and helps to reduce the program execution time by making it easy to execute the server-side remote code.

Acknowledgement

This study was supported by the BK21 FOUR project funded by the Ministry of Education, Korea (No. 4199990113966). This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (No. NRF2019R1A2C 2005099), and Ministry of Education (No. NRF2018R1A6A1A03025109).

References

[1] J. Park, M. M. Salim, J. Jo, J. C. S. Sicato, S. Rathore, and J. Park, “CIoT-Net: a scalable cognitive IoT based smart city network architecture,” *Human-centric Computing and Information Sciences*, vol. 9, article no. 29, 2019. https://doi.org/10.1186/s13673-019-0190-9

[2] S. B. Calo, M. Touna, D. C. Verma, and A. Cullen, “Edge computing architecture for applying AI to IoT,” in *Proceedings of 2017 IEEE International Conference on Big Data (Big Data)*, Boston, MA, 2017, pp. 3012-3016.

[3] X. Chen, Q. Shi, L. Yang, and J. Xu, “ThriftyEdge: resource-efficient edge computing for intelligent IoT applications,” *IEEE Network*, vol. 32, no. 1, pp. 61-65, 2018.

[4] C. Doukas and I. Maglogiannis, “Bringing IoT and cloud computing towards pervasive healthcare,” in *Proceedings of 2012 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, Palermo, Italy, 2012, pp. 922-926.

[5] T. Fujii, T. Toi, T. Tanaka, K. Togawa, T. Kitaoka, K. Nishino, N. Nakamura, H. Nakahara, and M. Motomura, “New generation dynamically reconfigurable processor technology for accelerating embedded AI applications,” in *Proceedings of 2018 IEEE Symposium on VLSI Circuits*, Honolulu, HI, 2018, pp. 41-42.

[6] A. Botta, W. Donato, V. Persico, and A. Pescape, “Integration of cloud computing and Internet of Things: a survey,” *Future Generation Computer Systems*, vol. 56, pp. 684-700, 2016.

[7] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of things (IoT): a vision, architectural elements, and future directions,” *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645-1660, 2013.

[8] A. Ahmed, and E. Ahmed, "A survey on mobile edge computing," in *Proceedings of 2016 10th International Conference on Intelligent Systems and Control (ISCO)*, Coimbatore, India, 2016, pp. 1-8.

[9] L. Hou, S. Zhao, X. Xiong, K. Zheng, P. Chatzimisios, M. S. Hossain, and W. Xiang, "Internet of Things cloud: architecture and implementation," *IEEE Communications Magazine*, vol. 54, no. 12, pp. 32-39, 2016.

[10] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: a survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347-2376, 2015.

[11] X. Lin, J. Li, J. Wu, H. Liang, and W. Yang, "Making knowledge tradable in edge-AI enabled IoT: a consortium blockchain-based efficient and incentive approach," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 12, pp. 6367-6378, 2019.

[12] B. Karg and S. Lucia, "Towards low-energy, low-cost and high-performance IoT-based operation of interconnected systems," *Proceedings of IEEE 5th World Forum on Internet of Things (WF-IoT)*, Singapore, 2018, pp. 706-711.

[13] J. Ren, H.Guo, C. Xu, and Y. Zhang, "Serving at the edge: a scalable IoT architecture based on transparent computing," *IEEE Network*, vol. 31, no. 5, pp. 96-105, 2017.

[14] A. Gupta, R. Christie, and R. Manjula, "Scalability in Internet of Things: features, techniques and research challenges," *International Journal of Computational Intelligence Research*, vol. 13, no. 7, pp. 1617-1627, 2017.

[15] J. Duval and H. M. Herr, "FlexSEA: flexible, scalable electronics architecture for wearable robotic applications," in *Proceedings of 2016 6th IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, Singapore, 2016, pp. 1236-1241.

[16] P. Mantovani, G. Di Guglielmo, and L. P. Carloni, "High-level synthesis of accelerators in embedded scalable platforms," in *Proceedings of 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, Macao, China, 2016, pp. 204-211.

[17] C. Yin, B. Zhou, Z. Yin, and J. Wang, "Local privacy protection classification based on human-centric computing," *Human-centric Computing and Information Sciences*, vol. 9, article no. 33, 2019. https://doi.org/10.1186/s13673-019-0195-4

[18] A. J. Perez, S. Zeadally, and N. Jabeur, "Security and privacy in ubiquitous sensor networks," *Journal of Information Processing Systems*, vol. 14, no. 2, pp. 286-308, 2018.

[19] L. Lei, Y. Kuang, N. Cheng, X. Shen, Z. Zhong, and C. Lin, "Delay-optimal dynamic mode selection and resource allocation in device-to-device communications—Part II: practical algorithm," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 5, pp. 3491-3505, 2016.

[20] J. Hou, T. Li, and C. Chang, "Research for vulnerability detection of embedded system firmware," *Procedia Computer Science*, vol. 107, pp. 814-818, 2017.

[21] Y. Xiao, Y. Jia, C. Liu, X. Cheng, J. Yu, and W. Lv, "Edge computing security: state of the art and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1608-1631, 2019.

[22] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80-84, 2017.

[23] M. D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali, "Cloud computing: distributed internet computing for IT and scientific research," *IEEE Internet Computing*, vol. 13, no. 5, pp. 10-13, 2009.

[24] D. Lee, J. Cho, and D. Park, "Efficient partitioning of on-cloud remote executable code and on-chip software for complex-connected IoT," in *Proceedings of 2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*, Kyoto, Japan, 2019, pp. 1-4.

[25] D. Park and J. Cho, "Cloud-connected code executable IoT device with on-cloud virtually memory controller for dynamic instruction streaming," in *Proceedings of IEEE International Conference on Cloud Computing and Big Data (CCBD)*, Shanghai, China, 2015, p. 29-30.

[26] D. Lee, H. Moon, S. Oh, and D. Park, "mIoT: metamorphic IoT platform for on-demand hardware replacement in large-scaled IoT applications," *Sensors*, vol. 20, no. 12, article no. 3337, 2020. https://doi.org/10.3390/s20123337

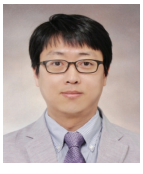
[27] G. McGrath and P. R. Brenner, "Serverless computing: design, implementation, and performance," *in Proceedings of IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, Atlanta, GA, 2017, pp. 405-410.
[28] L. Vojacek and M. Podhoranyi, "HPC based smart remote execution solution for modelling environmental issues," in *Proceedings of 2018 1st International Cognitive Cities Conference (IC3)*, Okinawa, Japan, 2018, pp. 242-245.
[29] M. Le, M. Song, and Y. W. Kwon, "Enabling flexible and efficient remote execution in opportunistic networks through message-oriented middleware," in *Proceedings of 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, Turin, Italy, 2017, pp. 979-984.
[30] Y. Fahim, H. Rahhali, M. Hanine, E. Benlahmar, E. Labriji, M. Hanoune, and A. Eddaoui, "Load balancing in cloud computing using meta-heuristic algorithm," *Journal of Information Processing Systems*, vol. 14, no. 3, pp. 569-589, 2018.

Dongkyu Lee **https://orcid.org/0000-0002-1318-6276**

He received the B.S. degree (Summa Cum Laude) in School of Electronics Engineering from Kyungpook National University in 2018. He is with the School of Electronics Engineering from Kyungpook National University, Daegu, Korea as an Integrated Ph.D. student. His research was focused on designing energy-efficient cloud-connected processors in VLSI chip level using on-demand remote code execution that can pervasively accelerate lots of services.

Moon Gi Seok **https://orcid.org/0000-0002-8159-9910**

He received the B.S. degree in electronics engineering from Korea University, Seoul, Korea in 2009, and the M.S. degree and Ph. D. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2011 and 2017, respectively. He was a post-doctoral researcher at KAIST and Arizona State University (ASU), Tempe, AZ, in 2018 and 2019. He is currently a research fellow at Nanyang Technological University, Singapore. His research interest includes multi-level modeling, simulation, and verification of system-on-chip (SoC) designs, and high-performance simulation methodology in various domains.



Daejin Park **https://orcid.org/0000-0002-5560-873X**

He received the B.S. degree in electronics engineering from Kyungpook National University, Daegu, Korea in 2001, the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2003 and 2014, respectively. He was a Research Engineer in SK Hynix Semiconductor, Samsung Electronics over 12 years from 2003 to 2014, respectively and have worked on designing low-power embedded processors architecture and implementing fully AI-integrated system-on-chip with intelligent embedded software on the custom-designed hardware accelerator, especially for hardware/software tightly-coupled applications, such as smart mobile devices, industrial electronics. He was nominated as one of Presidential Research Fellows 21, Republic of Korea in 2014. Prof. Park is now with School of Electronics and Electrical Engineering and School of Electronics Engineering as full-time assistant processor in Kyungpook National University, Daegu, Korea, since 2014. He has published over 170 technical papers and 37 patents.