Efficient Partitioning of On-Cloud Remote Executable Code and On-Chip Software for Complex-Connected IoT

Dongkyu Lee, Jeonghun Cho, and Daejin Park*
[1]School of Electronics Engineering, Kyungpook National University
*Correspondence to: boltanut@knu.ac.kr

***Abstract*—A program running on one processor can easily model the system and analyze power consumption and execution time. However, complex systems in which multiple processors interact are very difficult to model. Creating and simulating a simulation model would be effective for analyzing a complex system. However, the simulation model changes as the connection structure of the system changes. In this paper, we propose a framework that takes a connection structure and creates a simulation model automatically. This framework allows developers to easily create a simulation model by inserting the connection structures between the IoT systems. And we can find efficient part of on-cloud remote executable code and on-chip software in terms of power consumption or execution time.**

I. Introduction

Recently, IoT technology has been rapidly developing [1]. In IoT systems, edges consist of embedded systems. Each embedded systems is connected to the Internet and interacts with other embedded systems to perform certain services. There are two ways to run programs on an IoT system. One is to use the on-chip software and the other is to run the on-cloud remote executable code [2]. When the program is executed on-chip, there is no communication delay, but the program execution speed is slow and the number of services that can be provided is limited. However, running the program on-cloud has big communication delay, but the program execution speed is fast and the number of services that can provided is unlimited. Therefore, it is necessary to partition the code in order to operate the program efficiently according to the power consumption and execution time in IoT system. There is a lot of communication overhead between the embedded system and the cloud [3]. To reduce the overhead, the gateway mediates communication between the edge and the cloud and services simple things, such as translating to another communication protocol [4].

In the case of a simple system, we can easily calculate the power consumption and execution time necessary to perform a program [5]. However, simulation is easier than calculating power consumption and execution time in a complex system because simulation allows you to obtain outputs without calculations [6]. However, the simulation needs to be re-created when the connection structures of the system change. In this paper, we propose a framework that automatically creates a simulation model with connection structures and node configuration information. And we will consider the execution time and power consumption for result of the program. Using this framework, we can create an optimized simulation model and obtain information on the necessary power consumption and execution time to perform a program.

This study was supported by the BK21 Plus project funded by the Ministry of Education, Korea (21A20131600011). This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2014R1A6A3A04059410) and (2016R1D1A1B03934343) and (NRF-2018R1A6A1A03025109).

II. Proposed Architecture and Experimentation

Fig. 1 shows the layer structure of the proposed framework. The framework receives the physical connection of the system and the set of the program to be executed. The connection layer creates the conceptual connection between the nodes and gives connection information to the code generation layer, which has node information to create simulation code. With the node information and the conceptual connection received from the connection layer, the code generation layer generates a simulation model to simulate at the simulation layer.

The simulation layer receives the code, maps it to physical devices, then simulates it. Simulation results of the current conceptual connection are sent to the reporting layer. The framework returns to the connection layer to create another conceptual connections and repeats the sequences. The reporting layer shows the results of the power consumption and execution time of the program running on each conceptual connection. As a result of the reporting layer, we can find an appropriate ratio of the on-chip code and on-cloud code running on complex systems considering power consumption and execution time.

The connection layer receives the physical connection structure from the IoT system for which the simulation is desired. Within the same system connection structure, there are various ways to run the program. A typical method is to run the program on the edge and run the program on the server. The edge-side processing method can be advantageous in that fewer data transfer time is required, but it takes a long time to run the program. On the other hand, the server-side processing method is very fast in program execution but requires a large transfer time.

The connection layer creates conceptual connections using the edge-side processing method and server-side processing method as shown in Fig. 2. The conceptual connection shows the connections between each process in the IoT system. It allows the framework to know where the processes are executing and where each process communicates. The connection layer categorizes the processes according to where they are running.

978-1-5386-7789-6/19/$31.00 ©2019 IEEE

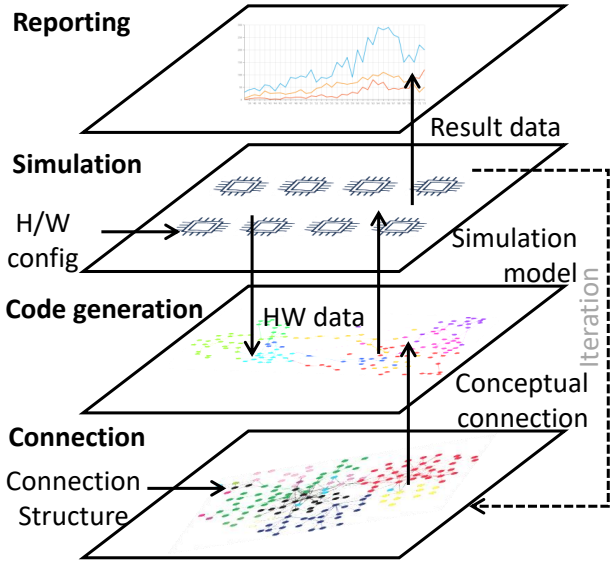

Fig. 1. The layer structure of the proposed framework

Then the server process and the edge process are connected to each other using the gateway process as shown in Fig. 2.

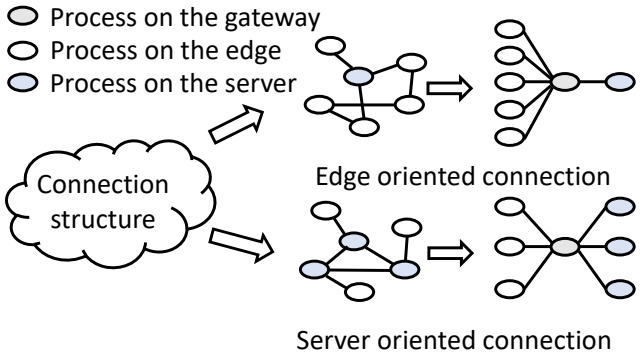

Fig. 2. The concept of the connection layer and code generation layer

Each node, including the edge, gateway, and server, has a static code part and a dynamic code part. The static code part has function information, so it does not change even if the connection structure changes. On the other hand, the dynamic code part has connection information. So it changes when the connection structure changes. When the code generation layer creates a simulation code, it changes the dynamic code part according to the connection structure. Using these characteristics, the code generation layer generates a set of the simulation code.

Similarly to Fig. 3, the simulation layer receives the set of the simulation code and implements it on each hardware. Then, the simulation layer simulates the received model and analyzes the power consumption and execution time of the connection model. The connections between the edge and the gateway and between the gateway and the server vary according to the connection structure and therefore belong to the dynamic code. The behavior of the components is modeled as a tiny instruction set simulator (ISS). The modeled tiny ISS belongs to a static code that does not change depending on the connection.

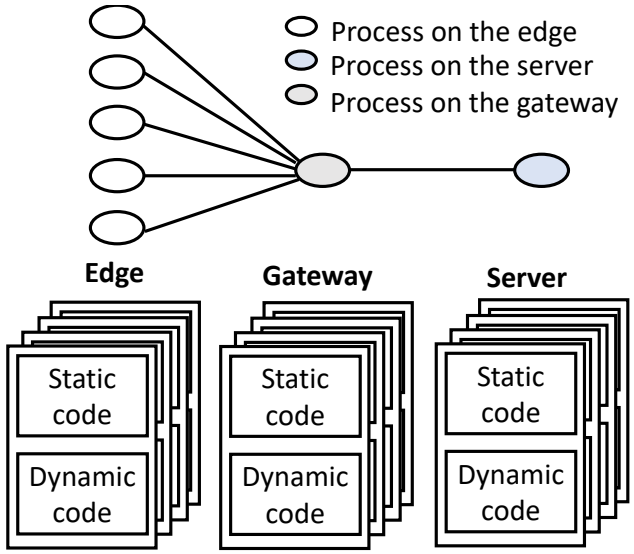

Fig. 3. The behavior of the simulation layer

Fig. 4 shows how the implemented simulation model works. The edges that want to communicate with the server request a service using a remote procedure call (RPC) to the gateway. The gateway executes RPC tasks and creates a thread for communication with the server, which has the code of the requested task. The server executes the code and returns the results to the gateway, and the gateway returns it to the edge.

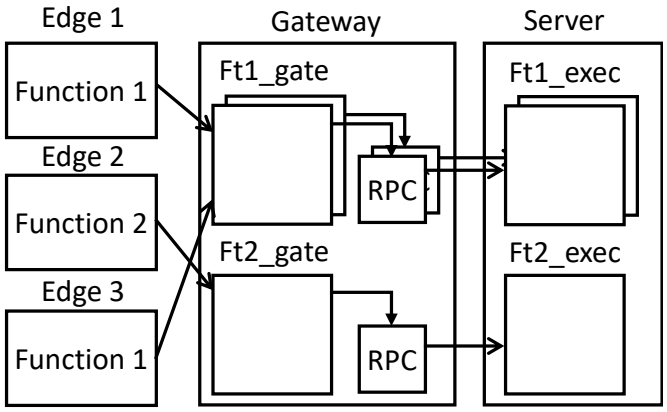

Fig. 4. The acts of the simulation model

Fig. 5 shows the results of the modeled simulation. In the case of a program that performs many operations on the edge, the edge performs more work and communicates less. It requires more power consumption at the edge, but it takes less time to communicate. On the other hand, a program that performs complex operations on the server can reduce operation time, but it takes more time to communicate than the edge-side program.

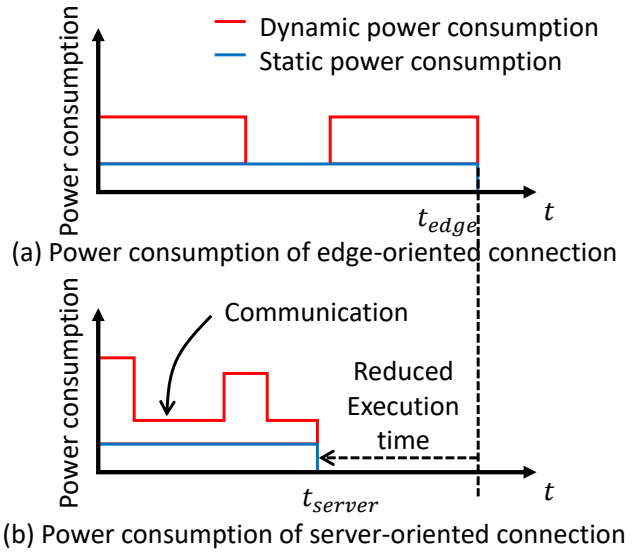

Fig. 5. The result of the modeled simulation

Each program has different characteristics. Some programs can have many complex operations, and other programs can transfer lots of data. To reduce the execution time and power consumption of a given system, this framework iterates the whole sequence and change the connection structure.

We assumed the edge to be a small processor and created an ISS to simulate the behavior of the tiny processor. Fig. 6 shows the concept of the tiny ISS. The code that is to be executed is stored in temporal storage inside of the ISS. When the set of the code is loaded into the temporal storage, the decoder fetches the code by the PC value and decodes it. The arithmetic unit then executes the code.

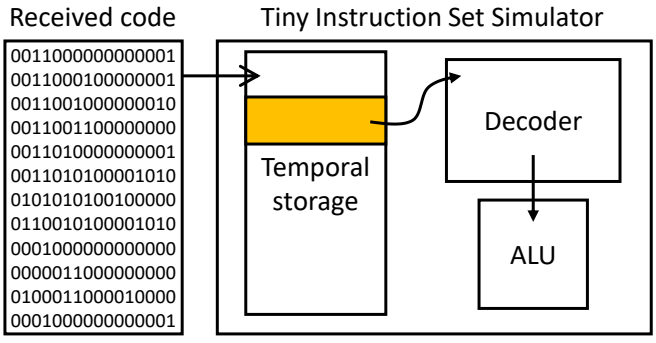

Fig. 6. The concept of the tiny instruction set simulator

The edge is made up of an ISS. In the case of edge-side processing, the ISS executes most of the program. On the other hand, in server-side processing, the ISS communicates with the gateway using an RPC that connects to a server that can run the program instead of the edge. Fig. 7 shows an example of RPC communication. By using an RPC, we do not have to worry about complicate settings for communication.

Inside the ISS code, we inserted the code for RPC communication as shown in Fig. 7. When the RPC function is called, the RPC program connects the gateway that the RPC server is running. The gateway receives the requests of the edge using the RPC then creates RPC processes to handle each request. The number of clocks per instruction and power consumption of each instruction are included in the ISS to calculate the program's execution time and power consumption.

```
case SND:
    // rpc_snd function
    clnt = clnt_create( server, GATEPROG, GATEVERS, "tcp" );
    if( clnt == (CLIENT*)NULL ) {
        clnt_pcreateerror( server );
        exit( 1 );
    }
    result = gat_snd_1( msg, clnt );
    if( result == (int*)NULL ) {
        clnt_perror( clnt, server );
        exit( 1 );
    }
    if( *result == 0 ) {
        fprintf( stderr, "SND error\n" );
        exit( 1 );
    }
    printf( "Send successfully!\n" );
    clnt_destroy( clnt );
    freq[ SND ]++;
    break;
```

Fig. 7. Example of the RPC communication code in ISS

Fig. 8 shows a part of the gateway code. We added a random sleep code in the RPC code to emulate a random connection time. The gateway mediates the connection between the edge and the server. To connect the server and the edge, the gateway creates a child thread that acts as an RPC client as shown in Fig. 8. Then, the child thread uses an RPC to request the server for the requested function at the edge. Finally, the server that acts as the RPC server performs the task. We also added a random sleep code in the server's RPC code to emulate a random connection time.

```
usleep(random*1000);          // Sleep random msec
printf( "Wait for %d ms\n", random*1000 );

// create thread
if( pthread_create( &th_id, NULL, t_snd, (void*)server ) != 0 ) {
    perror( "Thread create error: " );
    exit( 0 );
}
printf( "Gate success!\n" );

pthread_join( th_id, (void*)&rcv_result);
```

(a) RPC server code in gateway

```
clnt = clnt_create( (char*)server, MESSAGEPROG, MESSAGEVERS, "tcp" );
if( clnt == (CLIENT*) NULL ) {
    clnt_pcreateerror( server );
    exit( 1 );
}
printf( "create rpc success: \n" );
rcv_result = rpc_snd_1( msg, clnt );
if( rcv_result == (int*)NULL ) {
    clnt_perror( clnt, server );
    exit( 1 );
}
printf( "RPC success\n" );
if( *rcv_result == 0 ) {
    fprintf( stderr, "GATE error\n" );
    exit( 1 );
}

clnt_destroy( clnt );
```

(b) Child thread code that acts as an RPC client

Fig. 8. Example of the RPC communication code in gateway

To generate the simulation code automatically, we separated the generated code into a static code and dynamic code. Most of the code is static code. Because most of the code is independent of the connection structure, the dynamic code is associated with the connection structure. The code generator in the code generation layer receives the connection structure and generates an RPC code that constitutes as the dynamic code as shown in Fig. 9.

Fig. 10 demonstrates the implementation of the generated code to the actual hardware for simulation. The simulation

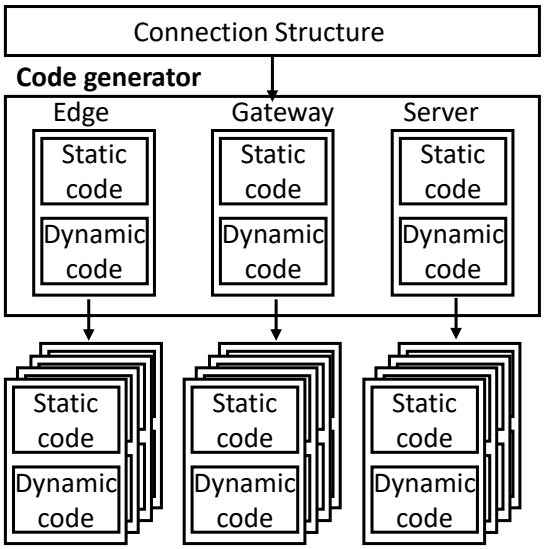

Fig. 9. The behavior inside of the code generator

code generated in the code generation layer is sent to the simulation layer. We implemented the edge code and the gateway code into the ARTiGO A820 IoT gateway and the server code into the ODROID cluster. In the ARTiGO IoT gateway, hundreds of ISSs are running to act as an edge, and the gateway code receives hundreds of requests to the edge and connects them to the necessary servers. As a result of the simulation, we were able to analyze the power consumption and execution time in the current connection structure. To find the point with the smallest power consumption or execution time, we simply changed the connection structure in the proposed framework and checked the results.

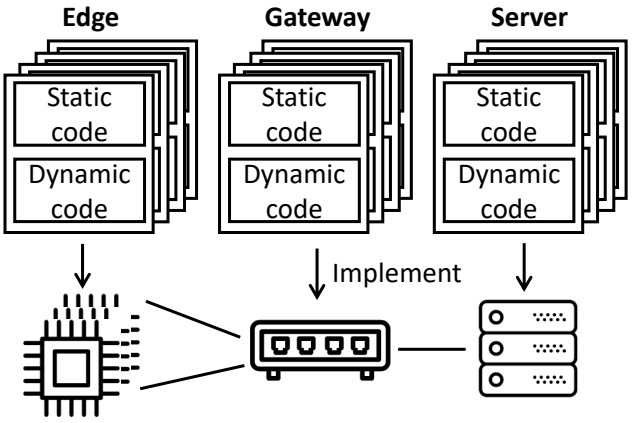

Fig. 10. Implement the generated code to the actual hardware for simulation

We create on-chip code-based simulation model and on-cloud code-based simulation model using proposed framework. And simulates a program with a lot of computation and a program with a lot of communication. We assumed that the execution time of the on-chip code in computationally intensive programs is 3 times greater than the execution time of the on-cloud code. And Fig. 11 represents the result of the simulation. In computationally intensive programs, the on-cloud code model has a shorter execution time than the on-chip code model because the server is faster than the edge. However, in communication intensive programs, the on-cloud model requires more communication than the on-chip model, so execution time is longer than the on-chip model.

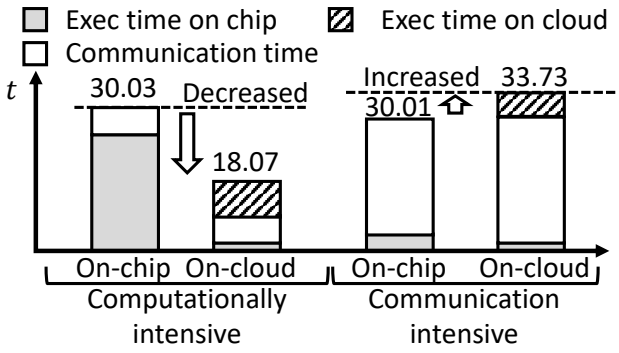

Fig. 11. Case study: the execution time of on-chip code and on-cloud remote execution code

III. Conclusion

This is our initial research for efficient partitioning of on-chip code and on-cloud code for complex-connected IoT. Analyzing programs that run on systems with complex connection structures can be difficult, but creating a simulation model is an effective way to make analysis easier. However, in order to find a proper connection structure, many simulation models must be created. In this paper, we propose a new framework that simulates a complex connected system in an IoT network. This framework creates simulation models for system connection and simulates them to find the power consumption of a given program. This is then repeated to find a connection structure that consumes less power and less execution time. Using this framework, we can easily find a connection structure that has minimal power consumption and execution time.

References

[1] 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.

[2] D. Park and J. Cho, "Cloud-connected code executable iot device with on-cloud virtually memory controller for dynamic instruction streaming," in *2015 International Conference on Cloud Computing and Big Data (CCBD)*, Nov 2015, pp. 29–30.

[3] G. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance," in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, June 2017, pp. 405–410.

[4] H. Chen, X. Jia, and H. Li, "A brief introduction to iot gateway," in *IET International Conference on Communication Technology and Application (ICCTA 2011)*, Oct 2011, pp. 610–613.

[5] T. Kubitza and A. Schmidt, "meschup: A platform for programming interconnected smart things," *Computer*, vol. 50, no. 11, pp. 38–49, November 2017.

[6] H. Cai, Y. Gu, A. V. Vasilakos, B. Xu, and J. Zhou, "Model-driven development patterns for mobile services in cloud of things," *IEEE Transactions on Cloud Computing*, vol. 6, no. 3, pp. 771–784, July 2018.