*electronics*

MDPI

*Article*

# On-Cloud Linking Approach Using a Linkable Glue Layer for Metamorphic Edge Devices

**Dongkyu Lee and Daejin Park ***

School of Electronic and Electrical Engineering, Kyungpook National University, Daegu 41566, Republic of Korea; dklee1215@knu.ac.kr

* Correspondence: boltanut@knu.ac.kr; Tel.: +82-53-950-5548

**Abstract:** As sensors operating at the edge continue to evolve, the amount of data that edge devices need to process is increasing. Cloud computing methods have been proposed to process complex data on edge devices that are powered by limited resources. However, the existing cloud computing approach, which provides services from servers determined at the compile stage on the edge, is not suitable for the metamorphic edge device proposed in this paper. Therefore, we have realized the operation of metamorphic edge devices by changing the service that accelerates the application in real time according to the surrounding environmental conditions on the edge device. The on-cloud linking approach separates the code for communication from the edge and server into a linkable glue layer. The separated communication code in the linkable glue layer is reconfigured in real time according to the environment of the edge device. To verify the computational acceleration of cloud computing and the real-time service change of the metamorphic edge device, we operated services that perform matrix multiplication operations with one process, two processes, and four processes in parallel on the edge–cloud system based on the on-cloud linking approach. Through the experiments, it was confirmed that the on-cloud linking approach changes the service provided in real time according to changes in external environmental data without changing the code built into the edge. When a square matrix operation with 1000 rows was loaded onto the proposed platform, the size of the code embedded into the edge device decreased by 8.88% and the operation time decreased by 96.7%.

**Keywords:** metamorphic edge device; embedded system; on-cloud linking

check for updates

**Citation:** Lee, D.; Park, D. On-Cloud Linking Approach Using a Linkable Glue Layer for Metamorphic Edge Devices. *Electronics* **2023**, *12*, 4901. https://doi.org/10.3390/electronics12244901

Academic Editor: Alexander Barkalov

Received: 26 October 2023
Revised: 25 November 2023
Accepted: 29 November 2023
Published: 6 December 2023

## 1. Introduction

A metamorphic edge device is a device that can provide different services in real time according to the surrounding environment, without recompiling the embedded application. Existing edge devices, which are locally deployed to collect data and perform simple computations, offer quick response times due to their close physical proximity to users [1]. However, edge devices are composed of low-performance hardware, resulting in slow computation speeds and limitations in the size of the code that can be embedded. In the existing application environment, edge devices require more branching code, which needs a larger memory space to cope with many situations occurring locally. A metamorphic edge device provides various services by replacing the operation of the application code without expanding the code memory or changing the embedded code.

Cloud computing and edge computing are methods to increase the insufficient computing power of edge devices. Cloud computing is utilized to compensate for the edge device's slow computation speed by transmitting data generated by the edge device to the server for processing. However, cloud computing creates communication overheads due to the significant physical distance between the data-producing device and the computation-performing device [2]. Therefore, it is crucial to examine the characteristics of the application and distribute computation appropriately between the edge device and the cloud server [3,4]. Metamorphic edge devices are also operated based on limited hardware like existing edge devices, so for fast calculations, application calculations are processed across the edge device and the server.

Existing edge–cloud systems provide only certain services to applications embedded in edge devices. To handle more environmental variables, a larger application code must be loaded onto the edge device, and to perform different operations, the running application code must be stopped and the code must be updated. Research has been conducted to optimize the code for hardware to provide more services within the limited memory of edge devices [5]. However, code optimized for hardware operates only on the hardware determined at the time of design, resulting in low maintainability. To design flexible programs for metamorphic edge devices, we separated the code optimized for hardware and the communication code for cloud computing.

This paper proposes an on-cloud linking approach for the flexible execution of metamorphic edge devices. The proposed on-cloud linking approach consists of an on-demand code execution method and a linkable glue layer generation method. For our on-demand code execution method, we separated the code in the application according to its computational intensity. The on-demand code execution operates services in the cloud as if the main program of the edge device were calling a function. The on-demand remote execution can provide services not embedded in edge devices, solving the memory constraints of edge devices. Furthermore, it allows the edge device to utilize the abundant resources of the cloud server in the background.

The linkable glue layer determines the server to which the on-demand execution code connects through a dynamic connection table and separates the code for service connection from the application code at the edge through an application programming interface (API). The proposed on-cloud linking approach changes the location of the server to execute the on-demand code without changing the embedded code by modifying the dynamic connection table of the linkable glue layer.

We conducted a case study evaluating the highly computationally intensive matrix multiplication operation to verify the on-demand code execution of the on-cloud linking approach. The matrix multiplication operation is an example that demonstrates the effectiveness of cloud acceleration of on-demand code because the amount of computation is higher than the amount of data transmission in an edge–cloud system. To verify the on-cloud linking approach, we loaded several types of matrix multiplication operation acceleration codes on the server. The built-in operation acceleration code is a code that distributes and executes multiplication operations with one processor, two processors, and four processors. In this paper's experiment, the edge device changes the server's service in real time at the linkable layer.

As a result of applying the on-cloud linking approach to the edge–cloud system, it was confirmed that the computation of edge devices can be accelerated in the cloud, like existing cloud computing. However, unlike existing cloud computing, applications on edge devices do not have code that is directly connected to the server through linkable glue code, and they call on-demand code as if calling code embedded in memory. Edge devices using on-cloud linking can perform more services with less memory than existing embedded software.

This paper used the on-cloud linking approach to create a metamorphic edge device that can change the cloud acceleration service it provides in real time depending on the environment without changing the embedded code. Section 2 describes the definition of metamorphic edge devices and the existing research, and Section 3 describes previous research on which this paper is based. Section 4 presents an on-demand remote code-execution-based environment for constructing an on-cloud linking approach. Section 5 describes the results of experiments conducted on matrix multiplication operations, which are examples of computationally intensive applications.

## 2. Background

A metamorphic edge device is an edge device that can dynamically change the services it provides in real time, and, unlike existing edge devices, it provides the same services regardless of changes in the surrounding environment. To handle various situations with the limited resources of edge devices, a metamorphic edge device employs an on-cloud linking approach to enable the use of the server's hardware in the cloud.

In an existing edge–cloud system, cloud computing is used to accelerate the computation of edge devices. Cloud computing is a method of reducing the execution time of the entire system by calculating data collected from edge devices through cloud applications [6]. Research was conducted on running an FPGA-based controller after calculating data collected from sensor-based edge devices in the cloud [7], and research on distributing and processing data collected according to a strategy in the heterogeneous cloud [8]. These cloud computing research studies only performed fixed-stage software load relocation to reduce application execution costs [9,10]. Research on changing the fault tolerance in cloud computing led to the construction of a real-time adaptive model by replacing weight information in fixed software [11]. Most cloud-computing-based adaptive systems use a limited connection change system that operates by reprogramming software at the design stage or replacing the weight of the fixed software. The on-cloud linking system proposed in this paper changes the running software flow in real time based on resource cloudification.

Cloudification is the process of moving a service to the cloud and virtualizing it for use over the internet. Current research on cloudification of various resources is based on cloud computing and includes hardware/infrastructure-as-a-service, platform-as-a-service, and software-as-a-service. Cloudifying a server's hardware resources allows edge devices to use more resources to speed up applications. The altered service code of the server enables edge devices to run applications without downloading changed services, effectively altering the application's code flow.

The proposed on-cloud linking approach requires computations to be executed on both edge devices and cloud servers. To set up a metamorphic edge device based on the on-cloud linking approach, micro processor unit (MPU)-level hardware capable of running an operating system (OS) is needed. For the communication protocol used to implement the on-cloud linking approach, a Cortex-A53 processor with 1 GB of RAM and 4 GB of flash memory allocated for the OS was used. In this paper, the edge device requires an ARM Cortex-A series MPU that can run an OS and a minimum amount of memory that can be equipped with an OS.

The metamorphic edge device assumes a continuous connection in an environment with a communication transmission speed of 10 Mbps or higher. To use a service on a server with hardware cloudification, data from the edge device must be transmitted to the server. The slower the communication environment, the greater the communication overhead of on-demand code execution. Compared to the existing embedded code, on-demand code execution adds service selection overheads due to the linkable glue layer and overheads due to data transmission. Therefore, on-demand execution code is effective when the execution time reduced by computational acceleration is greater than the execution time increased by communication overhead.

## 3. Related Work

Our previous work introduced an on-demand remote execution-based cloud execution platform to virtualize the server's memory and accelerator resources. Using on-demand remote execution, the edge device executes services on the servers as if calling a function from the main program on the edge device. With on-demand remote code execution, on-chip flash memory is virtualized and the service code is streamed to the edge in real time [12–14]. The executing service code is located on the server; thus, on-demand remote code execution can perform operations that are not built into the edge device. Furthermore, it allows the edge device to utilize the abundant resources of the cloud server in the background.

To enhance the computational speed of embedded devices, we studied an acceleration platform that utilizes on-demand remote code execution [15]. The proposed platform optimizes the program execution time by partitioning code based on their requirements: Code requiring a fast response time is executed on the embedded devices, while code demanding extensive computation is executed on the server. The remote code execution platform enables use of the server's resources on the cloud, enabling them to be executed at the edge. The program executed on this edge–cloud platform is divided into code that utilizes server resources and code that employs edge resources, thereby facilitating energy-efficient program acceleration.

Our existing research focused on dividing applications running in edge–cloud systems into embedded code and on-demand code and deploying them efficiently. Unlike existing cloud computing codes, the on-demand code proposed in previous research is easy to maintain because it consists of a part of a function rather than a service application. Going further than previous research, in this paper, we conducted research on changing on-demand code running in real time through an on-cloud linking approach.

Separating code based on the on-demand execution code enhances the independence of the edge and server code, enabling metamorphic program execution [16,17]. From a resource virtualization perspective, even if the code stored on the edge device remains unchanged, the behavior of the program executed at the edge can be changed if the code stored on the server is modified. Research has been conducted to provide intelligent services. To improve service performance in an edge-centric environment, a method for partially reconfiguring the hardware and firmware of the edge device according to the execution flow was proposed. Partial reconfiguration of edge devices is performed to provide metamorphic services. We conducted research on the real-time hardware reconfiguration of edge devices [18,19]. This real-time hardware reconfiguration research showed that the execution flow of an application can be predicted and that the hardware to be executed next to the edge device can be reconfigured in advance [20,21]. This process changes the service in real time by partially modifying the firmware of the edge device [22,23].

The presented research operates on edge-centric applications. The operation of the edge device can be expanded by connecting the edge and the cloud via research on distributing applications across multiple edges in the cloud using the over-the-air method [24–27]. From a memory virtualization perspective, if the code streamed from the server to the edge is changed, this results in different behavior on the edge device. In previous research, we accelerated data learning and classification on the edge device by cloudifying the server's FPGA and GPU using remote execution code.

## 4. Proposed Method

### 4.1. Collaborative Program Execution

Embedded devices execute all operations at the edge based on embedded code. Figure 1a illustrates an example of edge-intensive program execution. Data gathered from peripheral sensor devices are loaded into the memory of the embedded device. The loaded data are calculated by a service algorithm, and peripheral devices are controlled according to the results. Edge-intensive execution is not suitable for programs that require extensive computations, as all calculations are performed on low-performance embedded devices, resulting in a prolonged processing time.

Cloud computing, as shown in Figure 1b, serves as a method to compensate for the low computational performance of embedded devices. In the cloud-intensive computing approach, all data gathered from the embedded device are transmitted to the cloud server for processing and then relayed back to the embedded device. As all computations are carried out in the cloud, computation times are quicker than edge-intensive execution. However, additional time is required for data transmission. Furthermore, when an embedded device requests the server to execute an operation, the processor remains idle until the result is returned, leading to decreased resource utilization. Due to communication overhead, cloud-intensive execution exhibits faster speeds than edge-intensive execution when executing applications that necessitate more computation relative to data transfer.

Edge-intensive execution and cloud-intensive execution are efficient for simple application scenarios such as computation-only or communication-only programs. However, applications often contain a mix of parts in which data movement is high and parts in which calculations are concentrated. Therefore, the existing method of performing all calculations on one size is not suitable. Figure 1c shows collaborative execution, which involves distributing services between embedded devices and the cloud. Embedded devices process the collected data through a preprocessing stage. This processed data are then sent to the server for computation, and the results are returned to the embedded device for post-processing.

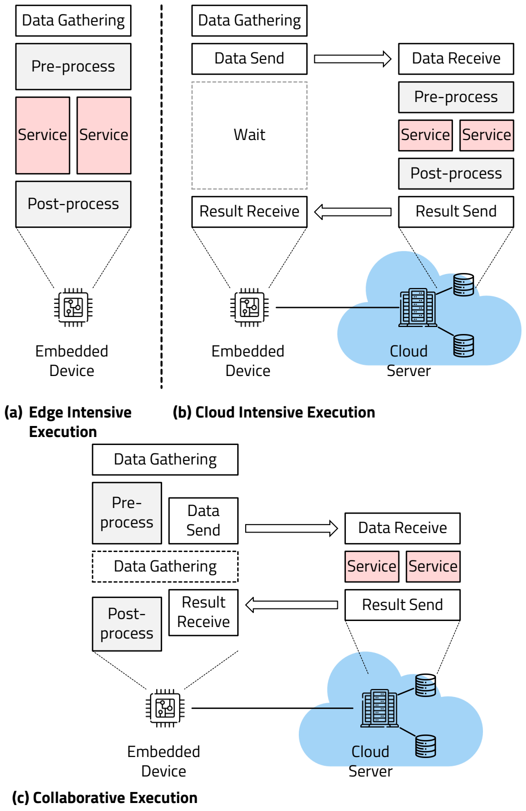


**Figure 1.** Software partitioning in a system with (**a**) edge-intensive execution, (**b**) cloud-intensive execution, and (**c**) a collaborative execution environment.

This paper discusses the software execution locations on edge devices and cloud servers in three environments for computation-intensive applications: edge-intensive execution, cloud-intensive execution, and collaborative execution. In edge-intensive execution, as shown in Figure 2a, all computations are performed on the processor at the edge after data are gathered from the sensor. However, due to the lower performance of the edge's processor, this method results in the longest execution time. In cloud-intensive execution, all computations are executed on the server in the cloud, and while the computation time is reduced because of the high-performance processors used by the server compared to the edge, data transmission takes longer. For cloud-intensive execution, as shown in Figure 2b, the edge device should maintain the data transmission channel using the processor while the server processes data.

Collaborative execution involves preprocessing data detected by sensors and then transmitting processed data to the cloud in bursts. Although preprocessing collaborative execution takes longer than cloud-intensive execution, it can reduce the amount of data to be transmitted. We outline the communication layer based on collaborative execution, enabling concurrent processing that can collect the next data while maintaining the communication channel, as shown in Figure 2c. This approach allows for more efficient use of resources and potentially faster overall execution times.

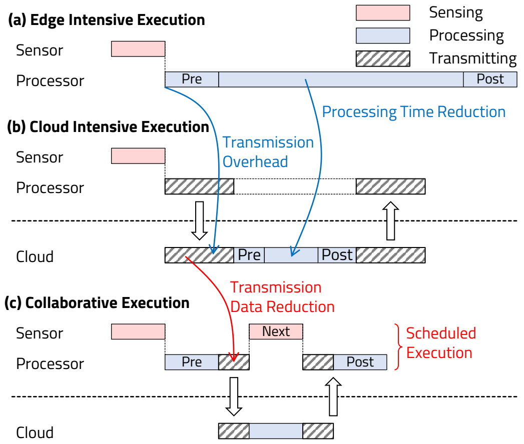


**Figure 2.** Software execution flow with (**a**) edge-intensive execution, (**b**) cloud-intensive execution, and (**c**) a collaborative execution environment for a computation-intensive application scenario.

### 4.2. Link-Layer Abstraction

In the existing server–client IoT system based on socket communication, the client code of the edge device and the service code of the server are paired, each containing connection information. Developers are required to create cloud-aware edge code and edge-aware server code, as shown in Figure 3a. To modify the service provided by the server, the connection code and the service code are to be changed and rebuilt. Similarly, to change the server to which the edge device connects, the link layer that provides the connection information in the client code must be modified and recompiled. In systems that provide fixed services, existing systems do not pose much of a problem. However, in systems in which the services provided by the edge device change depending on the surrounding environment, it is difficult to build a part by modifying only the necessary parts.

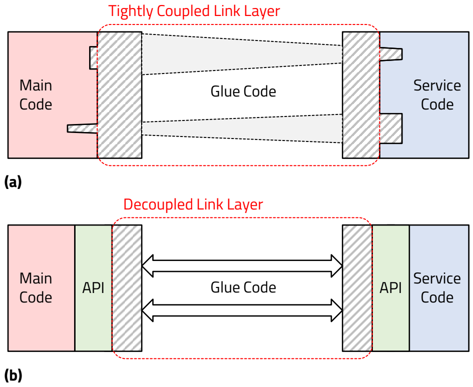


**Figure 3.** Concept of the connection code structure of an edge–cloud system: (**a**) existing code structure that requires a tightly coupled link layer design and (**b**) proposed on-cloud linking-based code structure.

We consider the communication layer to enhance the flexibility and scalability of services provided by the server–client system. Figure 3 shows the concept of the connection structure of the server–client system; Figure 3a represents the existing static linking approach. In the existing method, the connection layer between service codes is tightly coupled to the main code running on the edge and the service code running on the server. Due to the tightly coupled structure, the main code, service code, and glue code are designed as a single structure, and to modify one, all codes must be relinked. Figure 3b shows the proposed on-cloud linking-based approach. The existing link layer designs the glue code, considering the code to be connected, but in the proposed approach, the link layer is designed considering only the API. The API is a protocol for abstracting the link layer. The main code and service code designed by user use the API for communication, and the communication glue code between APIs is created by the communication layer generator.

Figure 4a shows the program structure, which is based on the existing socket-based static link layer structure. In the edge–cloud system, the embedded user application and the cloud service code are implemented with full awareness of each other. The user application that runs on the client is composed of an application part for processing data and a service call part for communication with the server. The client first preprocesses the data collected from the sensor, and then establishes a connection between the server and itself. The connection function is implemented by including the server's location information, which is predetermined by the user, into the client's application. The daemon running on the server is also designed with knowledge of the edge's system information. When the client request occurs, the daemon executes a fixed service code. The existing edge cloud structure requires mutual knowledge of each other's existence and necessitates the development of communication programs; thus, existing programs can only provide fixed services, limiting their flexibility.

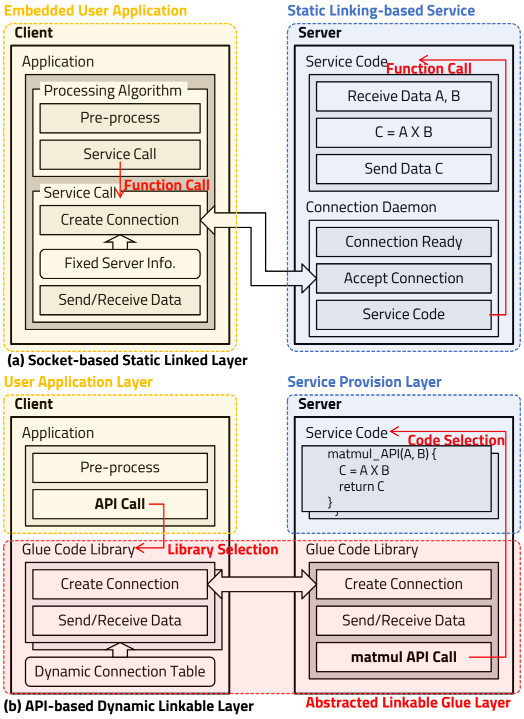


**Figure 4.** Edge–cloud execution system with (**a**) a socket-based static linked layer and (**b**) an API-based dynamic linkable layer.

Figure 4b shows the program structure using the proposed on-cloud linking approach. The entire program is composed of a user application layer that performs necessary operations on the edge device and calls remote code, a service provision layer that executes remote code on the server, and a linkable glue code layer that facilitates communication between the client and the server. The existing client server structure contains fixed connection information in the user application, preventing any changes to connection information on the client side. In contrast, the proposed method with the library structure stores connection information outside the user application and modifies the library operation of the communication layer based on the connection information, enabling real-time changes to the server service upon request. The user application calls the library using the API, and the invoked library establishes a connection to the server using current connection information.

The service structure of the conventional cloud server is such that the daemon processing and service codes for communication are compiled together, allowing only designated codes to be executed. Furthermore, because the communication code and the operation code are not separated, the service code is affected by the client's application operation. The proposed method separates the server's communication code and service code to ensure that service operations are not affected by the client application. The communication code receives data required for computation from the client in bursts and calls the service code via the API. In this proposed method, the service code invoked through the API can be modified by adjusting the connection configuration within the server.

Figure 5 illustrates the platform structure that reconfigures services executing as remote codes through the connection information of the abstracted link layer. The application code of the edge device consistently uses the same service API code, even if the provided service changes. The linkable glue code selects the service to be connected based on separately stored service selection information. Connection information for services executed by the API is stored in the link layer library. The link layer modifies connection information in real time according to the surrounding environments of the edge device, and the service called by the API changes accordingly. The server connected to the link layer has the service codes corresponding to the API connection information. When the remote execution request is entered through the API from an edge device, the service provider supplies the execution code determined by the code manager inside the server in the corresponding service code.

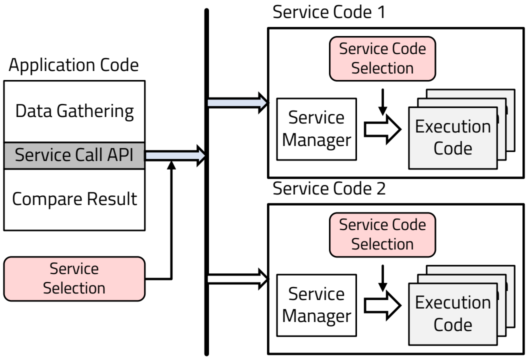
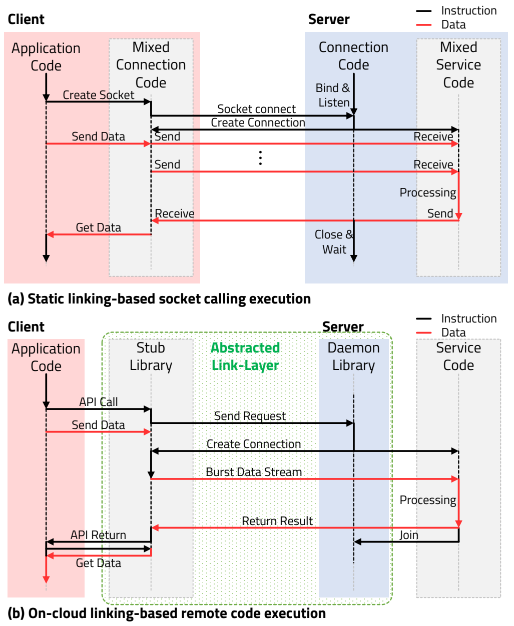


**Figure 5.** Link layer abstraction-based execution flow reconfiguration platform.

### *4.3. Remote Code Execution-Based Server–Client Environment*

Remote code execution virtualizes a server's resources, enabling the program running at the edge to utilize the server's resources as if they were local. Figure 6a shows the hierarchical execution flow of the existing socket-based server–client program, while Figure 6b shows the hierarchical execution flow of the proposed on-cloud linking-based server–client program. In the socket-based program, the server opens a port and waits for the socket connection to receive requests from the client. The edge device connects a socket to the server using the server's IP and port number. Through the connected socket, the client transmits data to the server, and the server returns the results of the computed data to the client. Socket-based communication requires developers to synchronize the sequence for sending and receiving data between the server and client at the code level, necessitating the joint design of service code and application code.

The on-cloud linking-based program abstracts the connection information in the library and accesses it through the API in the application code. The server executes the daemon process in the abstracted link layer to receive requests from the client. The edge device calls the remote code library using the API according to the execution flow of the embedded application code. The invoked library uses an external connection table to determine which server to connect and the requested server receives data from the client and calls the service code through the API. The result of the service code is delivered to the client's library through the created connection, and the library returns the result to the application code through the API.



(a) Static linking-based socket calling execution

(b) On-cloud linking-based remote code execution

**Figure 6.** Hierarchical calling sequence between the client and the server with (**a**) a static linking-based approach and (**b**) an on-cloud linking-based approach.

The remote execution-based platform efficiently reduces the execution time in environments where data production and computation are separated and require lots of computations. We loaded and executed the matrix multiplication operation, which requires large-scale data operations in the machine learning field, onto the proposed platform. Algorithm 1 shows the pseudocode of the matrix multiplication calling program executed on the embedded edge device. The application code, as denoted by *main* in Algorithm 1, interacts with the user and communicates with the link layer through the API. The executed *main* function calls *abstracted_connection*, a connection function of the linkable glue layer library. The *abstracted_connection* process is a stub code on the client side among the abstracted link layers in Figure 4, and is a code that is automatically generated based on the connection information, providing remote execution services.

The called *abstracted_connection* process configures a connection to the server using external connection table information in the background of the main application. When the connection to the server is established by the linkable glue layer library, the main application code requests operations on the collected data through the API named *matmul*. The background process that creates the connection to the server transmits matrix data $A$ and $B$ that were received through the main application's API at the server for on-demand execution. When the calculation on the server is complete, the link layer receives the result $C$ and delivers it to the application code.

Algorithm 2 shows the pseudocode for the remote code-execution platform running on the server. The server manages the connection through the daemon process generated in the linkable glue layer library, as denoted by *abstracted_connections*. On receiving the remote code execution requests, the server initiates a background process for the data transmission stream. The background process *abstracted_connections* receives data $A$ and $B$ from the edge device. This background process establishes a connection to the client's data transmission channel to receive the necessary data for computation. If the connection information in the server's linkable layer points to *matmul_body*, the server provides *matmul_body* as a service when a request for on-demand code execution is received. The result $C$ calculated by *matmul_body* is sent back to the edge by the linkable layer.

**Algorithm 1:** Application code executing on the client for a case study of matrix multiplication with computationally intensive operations

```
Input  : $A, B \in M_n(F)$
Output : $C \in M_n(F)$
% Application code
Function main
    Create abstracted_connection
    Gather data $A$ and $B$
    $C$ = matmul ($A, B$)
% Linkable glue layer library
Process abstracted_connection
    Get connection information
    Create connection and wait request
    if Call matmul then
        Send( $A, B$ )
        Receive( $C$ )
    Destroy connection
```

**Algorithm 2:** On-cloud code executing on the server for a case study of matrix multiplication acceleration and service selection

```
Input  : $A, B \in M_n(F)$
Output : $C \in M_n(F)$
% Linkable glue layer library
Process abstracted_connection
    Wait request
    Create connection
    Receive( $A, B$ )
    if Request Accel then
        matmul ← matmul_accel
    else
        matmul ← matmul_body
    $C$ = matmul($A, B$)
    Send( $C$ )
% Service code
Function matmul_body($A, B$)
    $C = A \times B$
    return $C$
Function matmul_accel($A, B$)
    Implement Accelerator
    $C = A \times B$
    return $C$
```

To accelerate remote code execution services and increase the utilization of the edge device's resources, we present the construction of an edge–cloud parallel processing platform. Figure 7 shows the flow of the remote code execution within the proposed parallel processing platform. In previous platforms, the client requests remote code execution and subsequently enters a waiting state until the results from the service are returned. However, in the parallel processing platform, the client separates the communication layer and the computation layer and performs computation on the edge device, enabling computations on the edge device and data communication with the server concurrently. The separation of computation and communication facilitated by the abstracted link layer allows edge devices to collect and transmit data simultaneously.

The background communication process interacts with the server based on the API connection information and transmits data required for computation. The server within the parallel processing platform processes data received from the client's background communication process by distributing them across the server's resources. When the daemon process operating on the server receives a remote code execution request from the client, it allocates available resources, creates multiple executable processes, operates the data, and transmits the results back to the client.

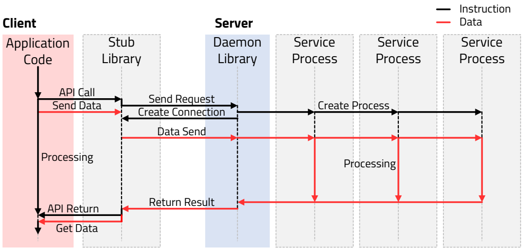


**Figure 7.** Remote code execution-based parallel processing platform execution flow.

## 5. Experiment and Measurement Results

### 5.1. OpenAPI-Based Remote Execution Platform

In this paper, we used a Raspberry Pi 3 as an edge device to collect data and request remote execution code, and as well as a desktop using Ubuntu OS as a server to provide a remote execution code service. Table 1 shows the specifications of the edge device and the desktop used to configure the platform. Raspberry Pi 3 uses an ARM Cortex-A53 CPU and has 1 GB of memory. The desktop that acted as a server uses an Intel i5-7600 CPU and has 16 GB of memory.

**Table 1.** Specification of the edge device and the server.

| | Edge | Server |
|---|---|---|
| Device Name | Raspberry Pi 3 | Desktop |
| CPU | ARM Cortex-A53 @1.2 GHz | Intel i5-7600 @3.5 GHz |
| Memory Size | 1 GB | 16 GB |

To verify the operation of the remote code execution platform, a square matrix multiplication with $n$ number of rows was operated using one edge device, the remote code execution platform, and the remote code parallel execution platform. A matrix multiplication operation is representative of an operation-intensive process in which the amount of calculations is large compared to the amount of data being transmitted. We adopted the matrix multiplication operation to clearly demonstrate the cloud acceleration of the remote on-demand executed code. Equation (1) represents the time required to calculate a square matrix with $n$ rows in an edge device. If $t_{CPU}$ represents the time taken to perform one multiply and accumulate (MAC) operation on an edge device, then $t_{edge}$, the time required for the matrix multiplication operation, is equivalent to the time taken to perform $n^3$ MAC operations.

$$t_{edge} = \sum_{1}^{n^3} t_{CPU} \tag{1}$$

For remote code execution in the abstracted communication layer, data must be transmitted from the client to the server. If $t_{tran}$ is the time required to transmit one unit of data and $t_{acc}$ is the time to perform the MAC operation on the server, then the total execution time $t_{remote}$ is the sum of the time taken to transmit $3n^2$ data and the time taken to perform $n^3$ MAC operations, as shown in Equation (2). Programs executed on the remote code execution platform require that the time reduction achieved through computational acceleration surpasses the time required for data transmission. Therefore, the larger the volume of data required for the MAC operation, the more effective it becomes.

$$t_{remote} = \sum_{1}^{3n^2} t_{tran} + \sum_{1}^{n^3} t_{acc} \tag{2}$$

In the parallel processing platform, the MAC operation is distributed and executed into multiple processes. Equation (3) represents the execution time of remote code execution based on the parallel processing platform. On the parallel processing platform, because the MAC operation is distributed, the computation time is reduced by $N$ processes executing simultaneously. However, additional time $t_{pre}$ is required to distribute and deliver data and to aggregate the data computed in each process.

$$t_{par} = \sum_{1}^{3n^2} t_{tran} + \sum_{1}^{n^3} t_{acc}/N + t_{pre} \tag{3}$$

### *5.2. Application Execution Time*

To evaluate the proposed parallel processing platform, a multiplication operation was performed on a square matrix with 1000 rows. MAC operations require a large number of operations relative to the amount of data, thus confirming the high performance of the parallel processing platform. Figure 8 shows the execution time when MAC operations are carried out in parallel on the edge device versus when they are executed on the parallel processing platform. When the MAC operation with 1000 rows was conducted with one process on Raspberry Pi 3, which has four logical cores, it took an average execution time of 121.59 s. On edge devices, MAC computations with two processes took an average of 54.42 s, and MAC computations with four processes took an average of 28.38 s, as shown in Figure 8. Due to the disparity in computational power between the server and the edge device, the execution time on the parallel processing platform using only one process was significantly reduced to 3.95 s, sharply contrasting the execution time on the edge device.

The execution time of the matrix multiplication acceleration platform, which is based on remote execution, varies depending on the services supported by the server and the resources it utilizes. Matrix multiplication is performed by incrementing processes on the parallel processing platform executing on the server composed of an Intel i5 CPU with four logical cores. The remote execution platform using one process took an average of 3.95 s to execute the matrix multiplication, the platform using two processes took an average of 2.10 s, and the platform using four processes took an average of 1.46 s.

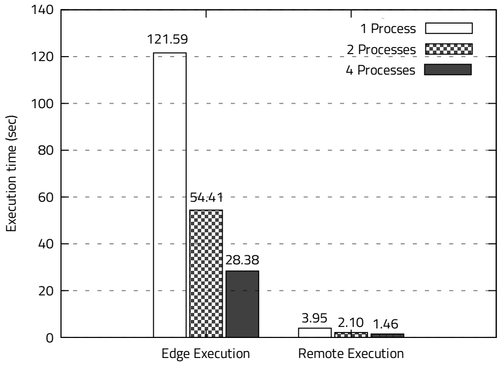


**Figure 8.** Execution time of the matrix multiplication on the proposed parallel processing platform.

Figure 9 shows the time taken for data preparation and transfer when the program was run on the remote parallel processing platform, and the time when the computation was actually performed on the server. The communication time required to prepare data at the edge of the parallel processing platform and transmit the data to the server was on average 0.67 s, regardless of which service is running. When executing the program in a server environment that does not use the proposed platform, the total execution time is 3.02 s. When the program is executed using the remote execution on the proposed platform, the request managing time is added at $t_{pre}$, as shown in Equation (3), so the computation time increases to 3.293 s and the execution time, including the data transfer time, is 3.95 s. However, in services that use multiple processes, the data transfer time and remote code request managing time are the same, but the matrix multiplication time is inversely proportional to the number of processes $N$.

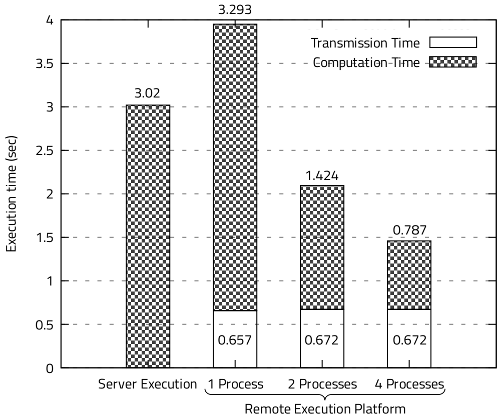


**Figure 9.** Computation time and transmission time of the remote parallel processing platform.

### *5.3. Code Flow Reconfiguration*

Figure 10 shows the contrast between the code configuration deployed on an existing edge device and the code configuration of the proposed platform. Code running on the embedded device is composed of data collection, computation, and result verification code. The proposed platform is divided into the edge application code and server service code, and these are interconnected by the link layer library. The application code is similar to the existing embedded code except for the code that is subject to remote execution. The data-processing code stored in embedded devices is abstracted into the API and represented in the application code. The API accesses the library of the link layer and executes services stored on the server. The service stored on the server consists of data-processing code extracted from the application code, service provision code that interacts with the link layer library and manages requests from the edge, and data transmission and reception codes. The matrix multiplication operation using the on-cloud linking method changes the server's service in real time with the linkable glue code generated by the connection information. We imported the services by the number of processes used by the matrix multiplication accelerator. The proposed edge device forms a foundation of a metamorphic edge device by changing the acceleration of matrix multiplication operations in real time.

Table 2 shows the sizes of the existing embedded code, application code, and service code in the proposed parallel execution platform for programs that perform matrix multiplication operations. In the traditional embedded device, the code that operates matrix multiplication and verifies the results is approximately 15.77 KB in size. In the proposed platform, the edge device's application code modifies the matrix multiplication function to the API for remote code execution. Compiling the modified application code and link layer library together reduces the code size by 8.88% to approximately 14.37 KB. Conversely, the size of service code on the server side is increased to about 18.46 KB due to the addition of code to manage requests from edge devices and transmit data. The proposed platform reduces the size of code stored in the edge device by providing the body of service code to be executed and stored on the server.

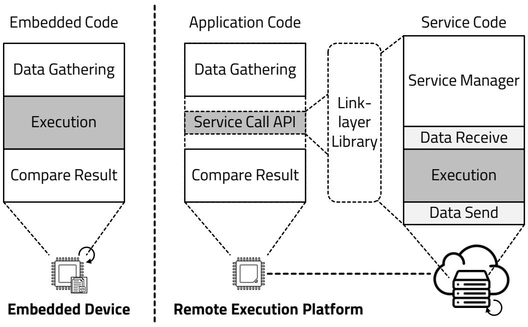


**Figure 10.** Differences in code composition between the embedded device and the proposed platform.

**Table 2.** Code size of the matrix multiplication program on the embedded device and the proposed parallel execution program.

| | **Embedded Device** | **Parallel Execution Platform** | |
|---|---|---|---|
| | | **Application Code** | **Service Code** |
| Code Size (KB) | 15.77 | 14.37 | 18.46 |

### *5.4. Discussion*

The proposed on-cloud linking method consists of cloud acceleration based on on-demand code execution and hardware cloudification based on the linkable glue layer. As a result of executing the matrix multiplication operation using the on-cloud linking approach, the execution time was reduced by 96.7% compared to when executing it on an embedded device. This experiment was conducted as a case study of a computationally intensive application, and it is expected that the decrease in execution time will be reduced when running an application with a low computational intensity. When an experiment was conducted to change the on-demand execution service in real time based on the linkable glue layer, it was confirmed that the service change and data transmission time was 0.657 s and the computation time was 3.293 s, resulting in communication overhead. However, the communication time overhead caused by the linkable glue layer can be considered a trade-off when considering the advantage of being able to change the service running in real time.

## 6. Conclusions

This paper proposed a real-time service reconfiguration platform through the on-cloud linking approach in an edge–cloud system. We evaluated on-demand code execution using on-cloud linking for a computationally intensive matrix multiplication operation application. The remote code execution structure of the proposed platform allows low-power embedded devices to utilize high-performance server resources as if they were local accelerators. The platform enhances the computational capabilities of edge devices. Based on the linkable glue layer of the on-cloud linking approach, edge devices can change the services that they call in real time. Existing embedded devices operate based on a pre-compiled code, which makes changing the code execution flow challenging. However, the proposed platform separates application and service codes through link layer abstraction, enabling changes to the code execution flow without modifying the code embedded in the edge device. To verify real-time service changes in metamorphic edge devices, a service that parallelizes matrix multiplication operations with one, two, and four processes, respectively, was executed in an edge–cloud system based on the on-cloud linking approach. Through the experiments, it was confirmed that the on-cloud linking approach changes the services that it provides in real time according to changes in external environmental data without changing the code embedded in the edge.

**Author Contributions:** D.L. contributed the implementation of software, validation, and original draft preparation; D.P. initiated the main conceptualization and estabilished the methodology; D.P. also supervised the entire manuscript writing as a corresponding author. All authors have read and agreed to the published version of the final manuscript.

**Funding:** This study was supported by the BK21 FOUR project (4199990113966), the Basic Science Research Program (NRF-2018R1A6A1A03025109, 10%) (NRF-2022R1I1A3069260, 10%) through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, and (2020M3H2A1078119) by Ministry of Science and ICT. This work was partly supported by an Institute of Information and communications Technology Planning and Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2021-0-00944, Metamorphic approach of unstructured validation/verification for analyzing binary code, 10%) (No. 2022-0-01170, PIM Semiconductor Design Research Center, 20%) (No. RS-2023-00228970, Development of Flexible SW-HW Conjunctive Solution for On-edge Self-supervised Learning, 50%). The EDA tool was supported by the IC Design Education Center (IDEC), Korea.

**Data Availability Statement:** Data used in the paper is contained within the article, and detailed data sharing will be conditionally applicable via email contact.

**Conflicts of Interest:** The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

## References

1. Rose, K.; Eldridge, S.; Chapin, L. The internet of things: An overview. *Internet Soc. (ISOC)* **2015**, *80*, 1–50.
2. Li, H.; Ota, K.; Dong, M. Learning IoT in edge: Deep learning for the Internet of Things with edge computing. *IEEE Netw.* **2018**, *32*, 96–101. [CrossRef]
3. Gai, K.; Qiu, M.; Zhao, H.; Liu, M. Energy-aware optimal task assignment for mobile heterogeneous embedded systems in cloud computing. In Proceedings of the 2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud), Beijing, China, 25–27 June 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 198–203.
4. Duan, S.; Wang, D.; Ren, J.; Lyu, F.; Zhang, Y.; Wu, H.; Shen, X. Distributed artificial intelligence empowered by end-edge-cloud computing: A survey. *IEEE Commun. Surv. Tutor.* **2022**, *25*, 591–624. [CrossRef]
5. Bakthavatsalam, G.; Mehata, K. A Case for Hybrid Instruction Encoding for Reducing Code Size in Embedded System-on-Chips based on RISC Processor Cores. *J. Comput. Sci.* **2014**, *10*, 411–422. [CrossRef]
6. Sharma, B.; Obaidat, M.S. Comparative analysis of IoT based products, technology and integration of IoT with cloud computing. *IET Netw.* **2020**, *9*, 43–47. [CrossRef]
7. Chen, Y. Research on programmable controller of construction machinery based on embedded system and cloud computing. *Microprocess. Microsyst.* **2021**, *82*, 103902. [CrossRef]
8. Du, X.; Tang, S.; Lu, Z.; Gai, K.; Wu, J.; Hung, P.C. Scientific workflows in iot environments: A data placement strategy based on heterogeneous edge-cloud computing. *ACM Trans. Manag. Inf. Syst. (TMIS)* **2022**, *13*, 1–26. [CrossRef]
9. Liu, H. Research on cloud computing adaptive task scheduling based on ant colony algorithm. *Optik* **2022**, *258*, 168677. [CrossRef]
10. Sefati, S.S.; Halunga, S. A hybrid service selection and composition for cloud computing using the adaptive penalty function in genetic and artificial bee colony algorithm. *Sensors* **2022**, *22*, 4873. [CrossRef] [PubMed]
11. Malik, S.; Huet, F. Adaptive fault tolerance in real time cloud computing. In Proceedings of the 2011 IEEE World Congress on Services, Washington, DC, USA, 4–9 July 2011; IEEE: Piscataway, NJ, USA, 2011; pp. 280–287.
12. Lee, D.; Seok, M.G.; Park, D. On-demand remote software code execution unit using on-chip flash memory cloudification for IoT environment acceleration. *J. Inf. Process. Syst.* **2021**, *17*, 191–202.
13. Rashid, A.; Chaturvedi, A. Virtualization and its role in cloud computing environment. *Int. J. Comput. Sci. Eng.* **2019**, *7*, 1131–1136. [CrossRef]
14. Alverti, C.; Psomadakis, S.; Karakostas, V.; Gandhi, J.; Nikas, K.; Goumas, G.; Koziris, N. Enhancing and Exploiting Contiguity for Fast Memory Virtualization. In Proceedings of the 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 30 May–3 June 2020; pp. 515–528. [CrossRef]
15. Guo, J.; Zhang, L.; Romero Hung, J.; Li, C.; Zhao, J.; Guo, M. FPGA sharing in the cloud: a comprehensive analysis. *Front. Comput. Sci.* **2023**, *17*, 175106. [CrossRef]
16. Lee, D.; Moon, H.; Oh, S.; Park, D. mIoT: Metamorphic IoT platform for on-demand hardware replacement in large-scaled IoT applications. *Sensors* **2020**, *20*, 3337. [CrossRef] [PubMed]

17. Lee, D.; Lee, S.; Oh, S.; Park, D. Energy-Efficient FPGA Accelerator With Fidelity-Controllable Sliding-Region Signal Processing Unit for Abnormal ECG Diagnosis on IoT Edge Devices. *IEEE Access* **2021**, *9*, 122789–122800. [CrossRef]
18. Samir, N.; Gamal, Y.; El-Zeiny, A.N.; Mahmoud, O.; Shawky, A.; Saeed, A.; Mostafa, H. Energy-Adaptive Lightweight Hardware Security Module using Partial Dynamic Reconfiguration for Energy Limited Internet of Things Applications. In Proceedings of the 2019 IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, 26–29 May 2019; pp. 1–4. [CrossRef]
19. Abdulmonem, M.H.; EssamEddeen, J.; Zakhari, M.H.; Hanafi, S.; Mostafa, H. Hardware Acceleration of Dash Mining Using Dynamic Partial Reconfiguration on the ZYNQ Board. In Proceedings of the 2020 32nd International Conference on Microelectronics (ICM), Aqaba, Jordan, 14–17 December 2020; pp. 1–4. [CrossRef]
20. Seyoum, B.; Pagani, M.; Biondi, A.; Buttazzo, G. Automating the Design Flow under Dynamic Partial Reconfiguration for Hardware-Software Co-Design in FPGA SoC. In Proceedings of the 36th Annual ACM Symposium on Applied Computing, New York, NY, USA, 22–26 March 2021; pp. 481–490. [CrossRef]
21. Moon, H.; Park, D. An Efficient On-Demand Hardware Replacement Platform for Metamorphic Functional Processing in Edge-Centric IoT Applications. *Electronics* **2021**, *10*, 2088. [CrossRef]
22. Novak, M.; Skryja, P. Efficient partial firmware update for IoT devices with lua scripting interface. In Proceedings of the 2019 29th International Conference Radioelektronika (RADIOELEKTRONIKA), Pardubice, Czech Republic, 16–18 April 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–4.
23. El Jaouhari, S.; Bouvet, E. Secure firmware Over-The-Air updates for IoT: Survey, challenges, and discussions. *Internet Things* **2022**, *18*, 100508. [CrossRef]
24. Yang, K.; Jiang, T.; Shi, Y.; Ding, Z. Federated Learning via Over-the-Air Computation. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 2022–2035. [CrossRef]
25. Zhu, G.; Huang, K. MIMO Over-the-Air Computation for High-Mobility Multimodal Sensing. *IEEE Internet Things J.* **2019**, *6*, 6089–6103. [CrossRef]
26. Arakadakis, K.; Charalampidis, P.; Makrogiannakis, A.; Fragkiadakis, A. Firmware over-the-air programming techniques for IoT networks-A survey. *Acm Comput. Surv. (CSUR)* **2021**, *54*, 1–36. [CrossRef]
27. He, X.; Alqahtani, S.; Gamble, R.; Papa, M. Securing over-the-air IoT firmware updates using blockchain. In Proceedings of the International Conference on Omni-Layer Intelligent Systems, Crete, Greece, 5–7 May 2019; pp. 164–171.

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.