electronics

MDPI

Article

BloSM: Blockchain-Based Service Migration for Connected Cars in Embedded Edge Environment

Srinidhi Kanagachalam, Khikmatullo Tulkinbekov and Deok-Hwan Kim *

Electrical and Computer Engineering Department, INHA University, Incheon 22212, Korea; ksrinidhi23@gmail.com (S.K.); mr.khikmatillo@gmail.com (K.T.)
* Correspondence: deokhwan@inha.ac.kr; Tel.: +82-32-860-7424

Abstract: Edge computing represents the future of computing paradigms that perform tasks near the user plane. The integration of blockchain with edge computing provides the added advantage of secured and trusted communication. In this paper, we propose a blockchain-based service migration by developing edge clusters using NVIDIA Jetson boards in an embedded edge environment, using containers and Kubernetes as a container orchestration capable of handling real-time computation-intensive deep learning tasks. Resource constraints in the edge and client movement are the proposed scenarios for service migration. Container migration due to mobile clients is integrated with blockchain to find a suitable destination, meta-based node evaluation, and secured data transfer in the connected car environment. Each service request migration takes, on average, 361 ms. The employed container migration method takes 75.11 s and 70.46 s to migrate application containers that use NVIDIA CUDA Toolkit. Finally, we evaluate the efficiency of blockchain to find the destination node through performance parameters such as latency, throughput, storage, and bandwidth.

Keywords: edge computing; containers; kubernetes; service migration; blockchain; container migration; Compute Unified Device Architecture (CUDA)

check for updates

Citation: Kanagachalam, S.; Tulkinbekov, K.; Kim, D.-H. BloSM: Blockchain-Based Service Migration for Connected Cars in Embedded Edge Environment. *Electronics* **2022**, *11*, 341. https://doi.org/10.3390/electronics11030341

Academic Editor: Luis Gomes

Received: 13 November 2021
Accepted: 19 January 2022
Published: 23 January 2022

1. Introduction

Blockchain and edge-computing paradigms emerged in IoT data-processing in recent years, with latency, security, and bandwidth advantages. These two technologies mainly have different aims. Edge computing relies on central clouds while moving some of the data computations to local nodes. On the other hand, Blockchain focuses on eliminating any central processing and fully relies on distributed computing. Since the first introduction, as the Peer-to-Peer(P2P) network powers Bitcoin [1], blockchain proved itself to be a reliable protocol for keeping data secure without third-party authentication. Blockchain shares the same control for all nodes in the network and requires all nodes to duplicate the data and store them forever. As everyone has the data, any misleading changes and unverified data can easily be ignored by the network. However, the blockchain protocol is designed to process small data, making it less effective in the edge-computing environment. However, there are specific use cases for the employment of blockchain in edge computing when security and privacy are top priorities [2,3].

In edge computing [4–7], the data computation job is divided into several smaller servers (edge nodes) located closer to the end users. Locally processing the data eases the job for data centers by sorting out insensitive data. Edge nodes are mainly designed to store local data and to run less heavy machine learning or similar applications. However, edge nodes have no computational resources such as data centers. These are mostly designed with embedded devices that use a limited computing power. There are cases when the edge node may not be able to complete the running application for unwanted reasons. This could be because the edge node may not have enough resources to complete the process or the connected device moves to another location, losing the edge node range [8].

For example, connected cars move quite quickly, so they easily get out of the edge node coverage range before the data processing is completed. The same example can be extended for user gadgets and other Internet of Things (IoT) devices to regularly interact with people. However, thanks to the overall design of the edge-computing environment, the IoT device is never lost within the network. The edge node range may be limited, but they are usually installed close to each other, covering the whole area as an edge network. This means that IoT devices can be disconnected from one node but remain in the network by connecting to another edge node [9]. Another problem occurs when the IoT device establishes a new connection to a different edge node. As each node runs independently of local applications, the new node does not know about the incomplete application in the previous node, so it may start the process anew. This scenario has two main drawbacks: (1) starting the process again creates more delay, and (2) dynamically moving IoT devices may lose the range again, resulting in a process that never ends. The service migration technique was developed for an edge-computing environment to deal with the above-mentioned situations. Instead of restarting the process, the new edge node migrates the existing application from the previous node. This is the ideal solution considering the IoT device requirements and system capabilities; however, service migration is not as easy as it sounds. First, migrating the service requires reliable and secure communication among edge nodes. Second, the migration latency should be minimized to satisfy the requirements of the IoT environment. Many researchers have been interested in implementing the best solutions for service migration, particularly focusing on the live migration of containers through dockers layered storage [10] and evaluating migration performance in a simulator tool [11].

Even though there are state-of-art solutions for service migration, they mostly focus on the migration process itself. On the other hand, enabling secure and reliable communication among edge nodes is an important issue to address, as mentioned above. Moreover, efficiently defining the source and destination nodes also considerably improves migration quality and speed. This research proposes a blockchain-based service migration technique, named BloSM, which is suitable for dynamically moving IoT environments such as connected cars. BloSM enables a blockchain network among edge nodes to share metadata for service migration. As the blockchain is the most secure distributed protocol, free of centralized authentication, it easily addresses security and privacy issues. Moreover, it also enables all nodes to see the migration requirements, incomplete applications, and migration status in real-time. The blockchain-based approach allows nodes to analyze the migration services before running them. Within the scope of BloSM, we claim four of the main contributions of this paper as follows:

- Blockchain-based metadata sharing shares the list of migration requests to all edge nodes.
- The Destination node's self-evaluation technique enables destination nodes to estimate their resource usage before migrating the service.
- Service migration, due to insufficient resources, enables load balancing and the efficient use of neighboring resources.
- Service migration due to client mobility maintains the service continuity, avoiding latency and connectivity issues.

2. Background and Motivations

2.1. Cloud and Edge Computing

At present, billions of devices are connected to the internet, requesting a zillion bytes of data. This increases the demand for computation; therefore, extensive research is being conducted every day. Edge computing, a recent advancement in computing, brings services and utilities closer to the user, reducing response times and latency [5,6]. It provides multiple advantages, including better bandwidth, support for mobile clients, and location awareness for real-life scenarios where the application response time is an essential factor and a fundamental requirement [7]. Researchers evaluated edge-computing performance, focusing on the network operational cost, and analyzing performance parameters, such as end-to-end latency, bandwidth utilization, resource utilization, and infrastructure cost [12].

Artificial intelligence (AI) technologies enable computational models to provide real-time solutions. AI-based edge computing has been proposed, which combines edge servers and deep learning models at the edge of the network [13,14].

As technology—mainly information, communication, and IoT—increasingly permeates society, achieving trusted access and secure authentication in the edge-computing plane becomes paramount. Blockchain technology addresses resource management and security issues before its deployment at the edge [3].

2.2. Blockchain

Blockchain technology has recently emerged in edge-computing systems, with a trusted protocol for data propagation. It was first introduced as a technology-powered Bitcoin and later as an entire cryptocurrency platform. Owing to its secure protocol for storing data in the public ledger, without the need for third-party authentication, many researchers have been working on adapting blockchain in various fields. Edge computing is another field in which blockchain offers great advantages in terms of data propagation latency and security. The term blockchain refers to the public ledger shared among multiple peer nodes, as shown in Figure 1.

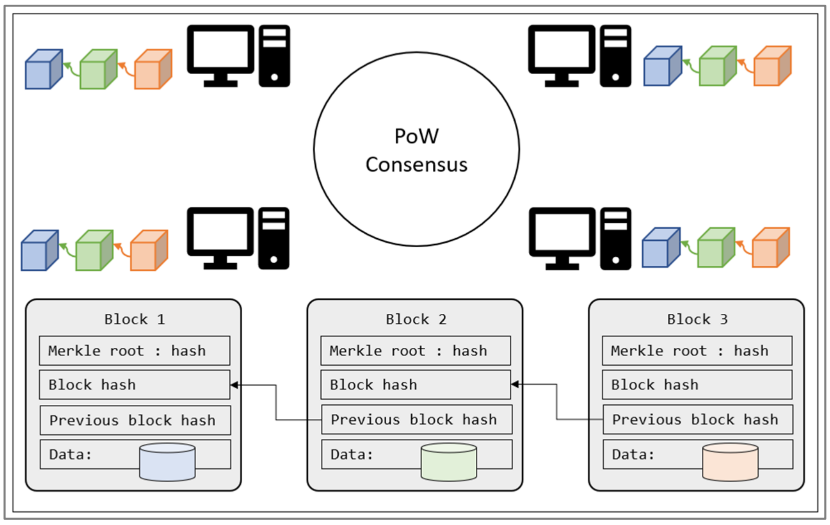


Figure 1. Blockchain overview.

Blockchain is a set of blocks that are linked to each other using the hash-referencing technique, as shown in Figure 1. Each block has four main attributes. The data contain the main data stored in the block. The data can be different based on the system to which they are applied. In the traditional Blockchain, data refers to the cryptocurrency transactions that are to be stored in the public ledger. The Merkle root hash value is generated in a binary tree form based on the list of transactions. This guarantees that each node in the network adds the exact same list of transactions in the block and provides integrity. Each new block also includes the previous block hash, which is the link for the chain. Based on all previously mentioned attributes, a unique hash is generated for the block and assigned as the fingerprint of the block. This hash is referenced in the next block, creating a single blockchain in the network. With the blockchain protocol, all nodes have the same right in the network, thus creating a distributed network. Alongside the same rights, all nodes share the same data throughout the network, enabling consistency.

In traditional blockchains such as Bitcoin, the proof-of-work (PoW) stands as the main consensus, providing security, reliability, and consistency. All nodes in the network agree on this consensus so that malicious actions can be detected and ignored within the network. According to PoW, a puzzle is given before each block creation and all nodes make an effort to solve the puzzle. The one to solve the puzzle first is verified by the other nodes (providing proof of its work) and writes a new block to the chain. To create a new block, the node is rewarded with some cryptocurrency (such as Bitcoin), so all nodes are highly motivated to participate in the PoW consensus. The PoW mechanism is highly effective in validating cryptocurrency transactions, but solving the puzzle is not an easy job. It takes about ten minutes until the new block is generated, which causes a delay for transaction approval. Moreover, the PoW consensus costs a lot in terms of energy, as all nodes continuously use their resources to solve the puzzle. These requirements and limitations mean that PoW is only applicable for high-computing environments with many GPU resources. Alternatively, researchers have come up with an energy-efficient solution named proof-of-authorization (PoA), which is more centralized compared to PoW. All the blockchain properties are still available with the PoA, but use a different consensus. The PoA allows one node to become the validator for the transactions and makes it responsible for creating new blocks. This allows transactions to be approved without an unwanted delay with the verification of the validator node. Due to its more centralized approach, PoA has not been adapted by cryptocurrency technology but is still effective in private organizations where all nodes are trusted and energy-efficient, which is the key factor, as in an edge-computing environment. Moreover, an edge-computing network is constructed of trusted edge nodes, so the size of the blockchain network is not the main factor in system security. This research extends the usages of the PoA-based Blockchain as a metadata-sharing solution to enable service migration in edge computing.

2.3. Related Work

Few studies have focused on service migration in the edge-computing plane, the emergence of blockchain, and its integration with the edge environment.

Wang et al. [15] proposed a dynamic service migration algorithm based on the Markov Decision Process (MDP) with users' known mobility pattern. In addition, W. Zhang [16] proposed a SEGUE technique focusing on Quality of Service (QoS) by considering the network state and the server state based on MDP. J. Lee et al. [17] proposed a QoS-aware migration considering the network cost, QoS, and resource utilization in the server. Machen et al. [8] conducted a live migration of Kernel-Based Virtual Machine (KVM) and Linux Containers (LXC) containers by implementing a layered framework for memory and filesystem sync by using checkpoints and restore functionalities and determined that LXC containers perform better in live migration than KVM containers. Studies on container technology reveal that it is resource-efficient, capable of running smaller instances that are easy to create, and supports faster migration. Ma et al. [10] pointed out that, with Checkpoint-Restore In User-Space (CRIU) functionalities, the whole container filesystem is transferred, irrespective of the storage layers, which leads to errors and a high network overhead. They also proposed a method through which the filesystem transfer size is reduced by using the docker's layered storage mechanism, which allows for the underlying storage layers to be shared before the migration and top layer storage to be shared after the migration begins. Bellavista et al. [11] proposed an architecture that migrates the service/data based on the application. Yin et al. [18] proposed a migration based on the service density of the current node and selected a destination based on the migration cost and user movement direction. In [19], Zhang et al. proposed a Q-network-based technique for task migration that learns the user's mobility pattern based on previous events, without knowing it in advance.

Blockchain, an upcoming technology, has recently gained significant attention in the field of the Internet of Things [20]. It is an immutable ledger where transactions take place in a decentralized manner [21], and several aspects of the integration of blockchain and edge computing have been explained in references [2,3]. A distributed and trusted authentication system based on blockchain and edge computing has been proposed to improve authentication efficiency [3]. Zhang et al. [22] proposed a blockchain-based secured-edge service migration focusing on security. Aujla et al. [23] designed a container-based data-processing scheme and blockchain-based data integrity management at the edge, to minimize link breakage and reduce the latency for Vehicle-to-Everything (V2X) environment. Van Thanh Le et al. [24] presented a Mobile Edge Computing (MEC) management based on blockchain in a swarm environment that provides service continuity in a secure manner, calculating the total migration time and analyzing the blockchain overhead.

2.4. Motivations

Significant challenges in edge computing include maintaining the network performance, Quality of Service (QoS) and service continuity despite mobile clients, providing security, and the deployment of low-cost fault-tolerant nodes. For example, autonomous vehicles, regarded as vast and time-critical applications, require the services to be kept close to the users to provide a quicker response time during the user movement [9]. In that case, moving the services to the next nearest edge servers (i.e., service migration) offer a feasible solution to the client, enabling service continuity. Most state-of-the-art solutions for service migration in edge computing, as discussed in Section 2.3, focus only on the migration process, considering the source and destination nodes. They also focus on the provision of security in service migration using blockchain [22]. However, finding the right migration destination remains an issue. Moreover, these solutions are designed for Intel or Advanced Micro Devices, Inc. (AMD)-based processors, which are mainly used by high-computing servers. Considering the environment outside the IoT world, resource-efficient and small-size embedded boards are more preferrable for installing on the sides of roads. However, most embedded devices come with Advanced RISC Machines (ARM)-based Central Processing Unit (CPU) architecture, where the existing service migration solutions may not be directly implemented.

To address these issues, this paper proposes a blockchain-based solution for service migration in edge computing called BloSM, focusing on real-time application with ARM-based embedded boards. It uses containers as a service carrier and Kubernetes for container orchestration in the edge environment. The edge clusters, consisting of edge nodes built with the Kubernetes platform setup, are more advantageous due to their minimal maintenance efforts. It also largely supports the migration of user applications without affecting the intracluster traffic. It reacts to possible failures and proactively prepares for migration. We implement service migration methods in Kubernetes, with a focus on providing a solution to the insufficient resources on the edge node by load-balancing between the available nearby nodes. We also focus on a container migration method for graphics processing unit (GPU)-based containers, suitable for mobile clients, in order to overcome the connectivity and latency issues. In addition, the integration of the container migration method with the blockchain to find the right destination node prevents further migration and provides secure communication between the source and destination nodes. The proposed method enables secured data distribution and trusted communication between edge nodes. In addition, using blockchain in service migration provides an easy solution for setting up source–destination node communication.

3. Preliminaries

As explained in the above sections, edge computing can be expanded to many implementations. Effective service migration solutions may vary depending on the specific environments. This research focuses on the location-friendly edge-computing environment, developed using the most optimal small-size embedded devices. The edge node requirements for building an edge-computing environment are listed in this section.

3.1. Edge Environment Setup

The edge-computing environment setup consisted of edge nodes (referred to as edge clusters from now on) built using NVIDIA Jetson development boards. The main reason for using these boards is the GPU availability in the small device. This gives the edge nodes the possibility of running machine-learning applications on embedded devices. In traditional methods, computers run a single environment on a single-host Operating System (OS), which results in the underutilization of resources. At present, container-based virtualization is one such technique for running multiple containers on a single OS, which improves performance and efficiency. Docker is a container technology that can easily package and distribute software based on its dependencies [25–27]. To create containers with a neural network service application, the NVIDIA Container Toolkit enables users to build and run GPU-accelerated Docker containers, as shown in Figure 2a. Docker enables TensorFlow GPU support; only the NVIDIA GPU drivers are required on the host machine.

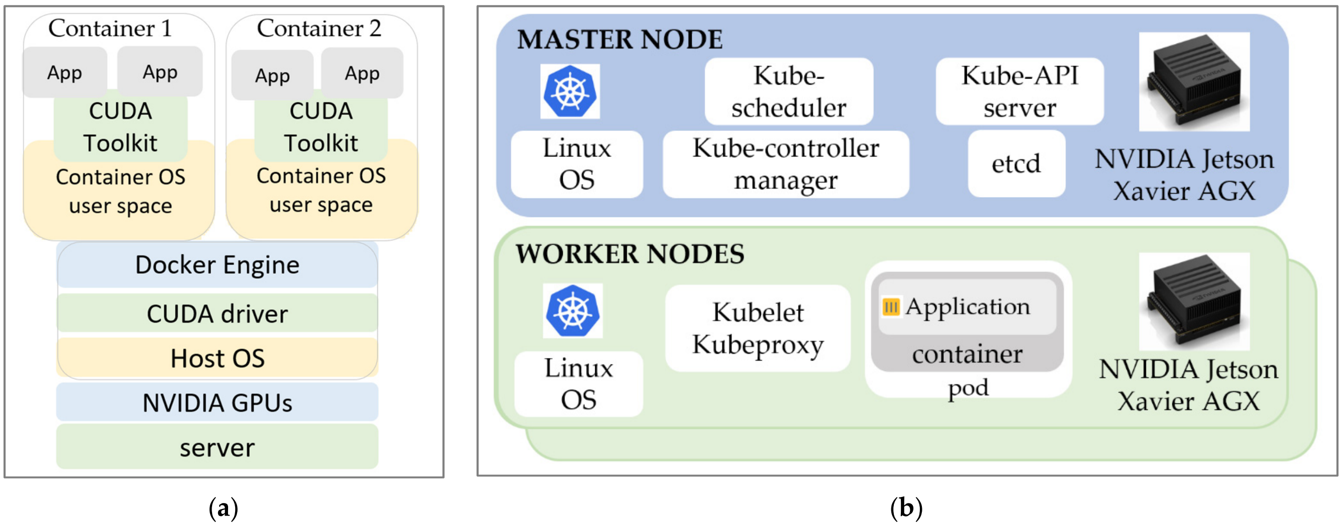


Figure 2. (**a**) GPU enabled docker container; (**b**) Edge cluster setup using Kubernetes.

Usually, edge nodes need to run several large-size applications simultaneously within a short time. To satisfy this requirement, each edge node includes more than one embedded board, where the hardware resources are orchestrated as one, forming an edge cluster. In other words, more than one embedded board work together to complete the job of a high-computing server. For resource orchestration purposes, Kubernetes is one of the most widely used orchestrators [28–30]. Kubernetes is an abstraction that allows for the efficient distribution of applications across a cluster of nodes. Each cluster can have one master node and any number of worker nodes. The components of the edge cluster using Kubernetes are shown in Figure 2b.

3.2. Deep Learning Applications

The applications that are used to perform service migration in our proposed edge server configuration focus on the connected car environment. Based on a real-time scenario, the following two deep learning applications were selected:

1. Driver Behavior Profiling using DeepConvLSTM [31].
2. Image recognition using MobileNetv2 [32].

The details of the application are mentioned in Table 1. For driver behavior profiling, the selected model was trained on the OCSLab dataset [33]. Among the 51 driving features that were acquired using the in-vehicle Controller Area Network (CAN) data bus, 15 features related to the engine (engine torque, friction torque, engine coolant temperature, etc.), fuel (fuel consumption, intake air pressure, etc.), and transmission (wheel velocity, transmission oil temperature, torque convertor speed, etc.) were selected. These 15 features were sent as input data with a window size of 40; hence, the total data size was 15×40. The output information contained the driver ID, execution time, confidence, and end-to-end time.

Table 1. Deep Learning application details.

Application	Model	Input Data Size	Output	Training Dataset	Testing Data in Our Proposed Environment
Driver Behavior Profiling	DeepConvLSTM	40 × 15	• Driver ID • Execution time • Confidence • End-to-end time	OCSLab	• JSON formatted • OCSLab dataset
Imagerecognition	MobileNetv2	224 × 224 × 3	• Object ID • Object name • Probability • End-to-end time	ImageNet	• JSON formatted • Encoded/decoded • Resized • Real-time images

For image recognition, the selected model was trained on the ImageNet dataset with 1000 classes. We input a real-time image after resizing it to 224 × 224 × 3 as per the requirement of MobileNetv2, and the output information contained the object ID, object name, confidence, and end-to-end time.

4. BloSM Design

4.1. Blockchain Network Setup

The employment of a blockchain network plays a crucial role in the proposed system. Traditionally, blockchain requires high computing power for stability and a larger network to achieve security, as mentioned in Section 2.2. However, the BloSM is developed for an edge-computing environment where edge nodes are constructed by embedded devices. Moreover, the edge-computing network is already constructed by trusted nodes. Based on these observations, BloSM extends the proof-of-authorization (PoA) consensus for the blockchain network. With the help of PoA, BloSM achieves the same security for both smaller and larger networks. Moreover, the application metadata can be verified without delays with a validator-oriented consensus.

When the connected client device is out of range, the source node (the edge node to which the device is disconnected) broadcasts the container metadata to the blockchain network. The metadata information will be available to all nodes, including the destination node (the edge node to which the device will be connected later). When a new user attempts to reconnect to the existing operation, the destination node can easily find the source node through the metadata information and migrate the user service. This process eliminates the need to search for a destination node during service migration. We should also consider the heterogeneity in the edge-computing environment. In some cases, the destination node may not have enough resources to handle migration. In this case, the destination node is required to migrate the user data to continue the service in the source node. To fulfill this purpose, the data structure of the metadata shared in the blockchain is important. The metadata should include sufficient information to evaluate the node resources and handle the evaluation results. Figure 3 shows an overview of the data structure of the blockchain network.

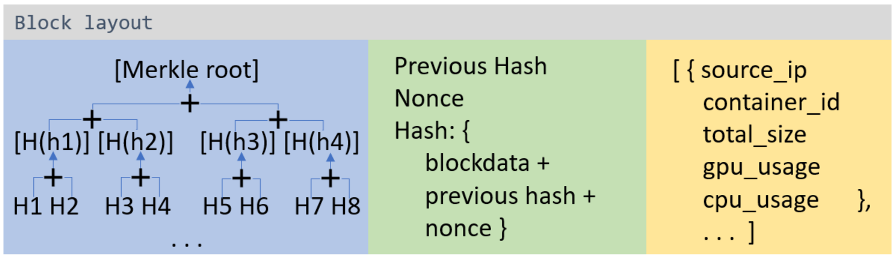


Figure 3. The blockchain data structure in BloSM.

Each block contains a set of transactions. In the BloSM scenario, each transaction contains metadata information about the container. First, two variables, *source_ip*, and *container_id,* are shared to find the source node. In addition, the metadata includes variables such as *total_size*, *gpu_usage*, and *cpu_usage* to enable the self-evaluation of the destination node. In this scenario, the list of transactions is stored as block data. Apart from transactions, the block also includes the previous block hash to create a link in the chain and the nonce value generated by the proof-of-work (PoW) procedure. A block hash is generated based on the information mentioned above. Another aspect of the block data structure is the Merkle tree. As the block is shared with all the nodes in the network, it is important to ensure integrity. All nodes are expected to add exactly the same order of transactions in the block. Thus, the Merkle root is generated based on all transactions inside the block to ensure consistency. By enabling the blockchain network, BloSM achieves the advantages listed below:

- *Availability.* Metadata information on the migrating application is easily broadcasted to the network, so all nodes have local real-time updates.
- *Privacy.* The shared metadata does not include information about processing data within the application but only abstract data, such as size and resource usage.
- *Utilization.* Blockchain eliminates cloud-based communication where service migration resources can be utilized.

4.2. Metadata Sharing and Node Evaluation

The process of sharing container metadata with the blockchain network is shown in Algorithm 1. For the prototype implementation, the procedure obtains information on the four resources as input. Input parameters include *CPU*, *GPU* usage, total *size* of the container, and *type* of application, as shown in lines 3–6. The procedure returns a *Boolean* value depending on the success of the operation. Line 9 indicates the network IP address of the source node. This value is used to access the source node to request the service migration. A new list is created based on the type of resource object, called *meta*. This list includes all the resource information that needs to be shared on the blockchain, as shown in lines 10 and 11. After initializing the metadata, the information is written to the blockchain as a new transaction with *the newTransaction*() method on Line 15. As lines 17 and 18 show, the procedure stops and returns as *false* if the operation is unsuccessful. When writing, the new transaction finishes successfully, and the procedure takes the number of confirmations from the blockchain network using *the blockchain confirmations*() method. The method waits until the propagation reaches at least 90% of the nodes in the network, as shown in lines 23 and 24. This is because the destination node needs to receive the metadata before the client connects to it. After the metadata are broadcast successfully, the destination node can perform an evaluation algorithm using the *metadata*.

As mentioned in the previous section, one of the main advantages of blockchain-based metadata-sharing in service migration is availability. This enables self-evaluation without migrating the application. Edge computing is constructed using multiple nodes with no specific hardware requirements. In addition, the present applications require different types of high-quality data processing. This leads to the applications being categorized into different types depending on the hardware resources they consume. It is clear that the source node is already capable of running a certain container before performing migrations. However, a question remains in the destination node: what if the destination node is not able to provide the highest resources, as in the source node? Traditionally, this problem is not considered a part of service migration; it is assumed that all edge nodes have the same resources. BloSM enables destination nodes to perform a self-evaluation even before migrating to the container, thus eliminating redundancy. To make this process possible, initially, the source node shares the container metadata with the blockchain network. All nodes can access the container metadata once they are propagated. When a destination node receives a request from a new user, it compares the user data with the container metadata on the blockchain. If they are found to belong to the same application, service migration is required, and the destination node starts the self-evaluation, as in Algorithm 2.

Algorithm 1 Sharing metadata on blockchain.

```
1: function WRITETOBLOCKCHAIN (cpu, gpu, size, type)
2:
3: Input:   cpu → CPU usage of the container
4:          gpu → GPU usage of the container
5:          size → Total size of the container
6:          type → Type of application
7: Output:  result → Boolean result of operation
8: procedure
9:     source_ip ← new getCurrentNodeIP()
10:    meta ← new List < ResourceType >
11:    meta.add(cpu, gpu, size, type, source_ip)
12:
13:    // Share on blockchain
14:    blockchain ← Blockchain.instance()
15:    status ← blockchain.newTransaction(meta)
16:
17:    if (!status.OK) then
18:        return FALSE
19:
20:    c ← blockchain.confirmations(status.ID)
21:    n ← blockchain.totalNodes()
22:
23:    while (c < n * 0.9) do
24:        c ← blockchain.confirmations(status.ID)
25:
26:    return TRUE
```

Algorithm 2 Destination node self-evaluation

```
1:  function PROCESS (meta, sys)
2:
3:  Input: meta → Container metadata
4:         sys → Node system information
5:  Output: request → Request to the source node
6:  procedure
7:      prob ← new List < Boolean >
8:      prob.add(meta.size > sys.memory(free))
9:      prob.add(meta.cpu > sys.cpu(free))
10:     prob.add(meta.gpu > sys.gpu(free))
11:     for app in sys.applications do
12:         if (app.type == meta.type) then
13:             prob.add(TRUE)
14:
15:     result ← TRUE
16:     if (prob.size == 3) then
17:         result ← FALSE
18:     else
19:         count ← 0
20:         for p in prob do
21:             if (p == FALSE) then
22:                 count ← count + 1
23:         if (count > 1) then
24:             result ← FALSE
25:
26:     return migrationRequest(meta.source_ip, result)
```

The algorithm takes the container *metadata* from blockchain and *sys* from system configurations as the input and returns a service migration *request* to the source node depending on the decision. The procedure creates a new list: *prob*, to store the comparison results. *size*, *cpu*, and *gpu* metadata resources are compared to the node resources one by one, as shown in lines 7–10. Depending on the comparison, the algorithm adds *true* or *false* values to the list, where true means that the node can provide this resource and false means the opposite. The next object of comparison is the application *type,* as shown in lines 11 to 13. The destination node checks all available applications in the node environment. If the application type matches the metadata type, then the *prob* list is extended with a *true* value. After finishing the evaluations, the next step is decision-making. The final decision is stored to the *result* variable based on different conditions, as shown in lines 15–24. The default value is *true*. It is updated to *false* in two conditions: (1) if the application type does not match the node capabilities (lines 16 and 7); (2) if more than one of the resources fail in comparison. Based on the final decision, the migration starts with the *migrationRequest* () function call (line 26). The function sends a request with a *Boolean* flag, which is the self-evaluation result. In the case of a *true* flag, the migration is performed to the destination node. Otherwise, the destination node cannot run the container locally; therefore, the migration is useless. In this scenario, the destination node will be able to transfer the user data to the source node to continue the operation instead of migrating the service. With the implementation of self-evaluation of a destination node, BloSM achieves the following:

- *Efficiency*: Service migration is performed only if the destination node can run the application.
- *Transparency*: Destination node knows the exact uses of the application before migration.

4.3. Service Migration Considerations

In the edge-computing plane, consider that the edge clusters $EC = \{ec_1, ec_2, \ldots, ec_n\}$ are connected and placed at a fixed location at a distance near the user plane to process user requests with reduced latency. There can be an 'n' number of internet-connected client devices $C = \{c_1, c_2, \ldots, c_n\}$ on the user plane requesting edge clusters for real-time information. Client devices generally have less computing power and are highly mobile. For a job that requires high computing power, the client device sends a connection request to the nearest edge server. Once the connectivity is established, the client sends real-time data to the edge cluster, and the edge executes the type of service that is requested. For a service request to be processed, the edge cluster should have sufficient computing resources, such as *memory*, *CPU*, and *GPU*. If the edge cluster has all these resources, then the request can be processed with reduced latency, and the client devices receive the response without any added delay.

The requirements for an edge cluster to process a request are as follows:

$$S = \{R_i, T_i\} \tag{1}$$

$$R_i = \sum\{Node_{memory}, Node_{CPU},\ Node_{GPU}\} \tag{2}$$

where R_i is the resource required by the edge cluster and T_i is the time taken by the edge cluster.

In addition, if the application image is available in the processing node, the image download time can be avoided. If not, then the node should spend additional time downloading the image. Until then, there is an additional waiting time when processing the request. The edge cluster process the requests and provides information as a service (IaaS) to the client. Each edge cluster is a small data center, and it has a coverage limit, up to which it can provide seamless service to the user. Problems occur when there are insufficient resources in the edge cluster, or when the client moves away from the coverage. The solution is service migration, which requires nearby destination nodes to which the services can be moved. We consider these two problems as two different scenarios and provide the best possible solution in Sections 4.4 and 4.5. The problem of finding the right destination node and secured information transfer in case of client mobility is addressed using blockchain. Figure 4 provides an overview of the proposed model for blockchain-based service migration.

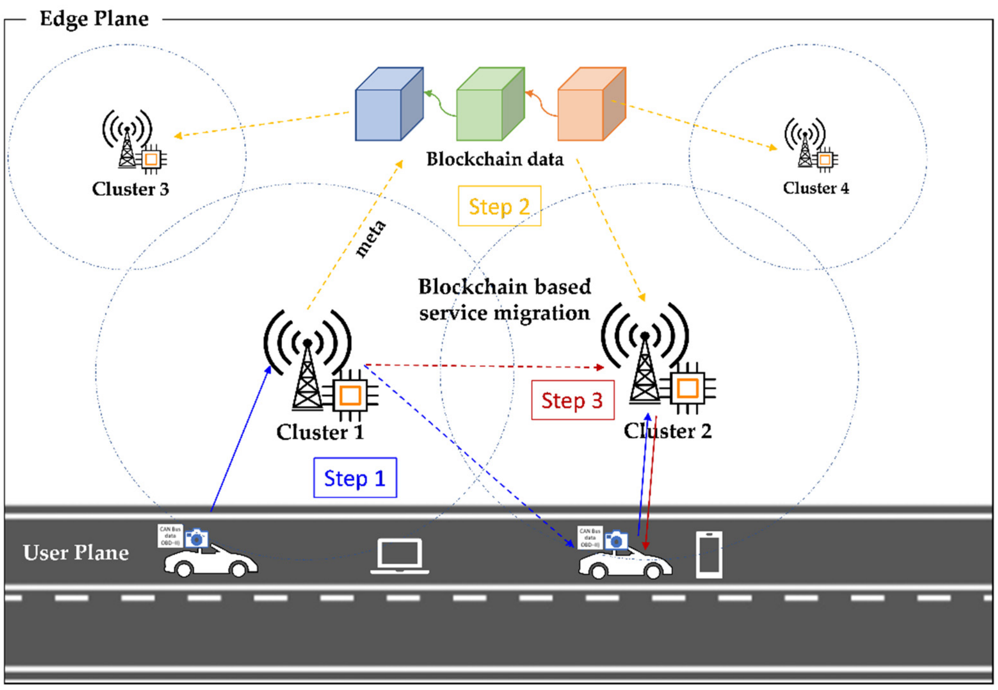


Figure 4. Overview of the proposed model for blockchain-based service migration.

4.4. Migration Due to Insufficient Resources

Insufficient resources occur due to the service density, which means the available resources on the edge cluster are utilized by many other client requests. If the resources in R_i of the edge cluster are insufficient to process the request, then the service request is scheduled to be migrated to the nearby edge cluster, avoiding the waiting time needed to process the request. The advantage of this is that it avoids the request-processing delay and maintains the QoS of the edge plane. In other words, it provides load-balancing between the edge clusters by using the resources in the surrounding edge nodes. As mentioned in Algorithm 3, for each request (REQ) that reaches the edge cluster, the IP of the master node is assigned as E_s. The requested type of service is read as REQ_{type}, and the resource required to process the request is REQ_{res} (lines 8–10). The remaining memory in the edge cluster is calculated using (3) and compared with REQ_{res}. If the required resources are higher than the remaining memory, then *Request_Migration(REQ)* migrates the request to the next nearest edge cluster. The remaining resources R_{rem} available in the edge cluster are calculated using the total available resources and the utilized resource.

$$R_{rem} = R_{tot} - R_{utilised} \tag{3}$$

4.5. Migration Due to Client Mobility

When the client connected to a particular edge server moves away from the coverage, the service container's migration is scheduled in order to maintain service continuity and avoid latency and connectivity issues. The service containers use NVIDIA CUDA Toolkit to run GPU-accelerated applications. These applications make use of both the GPU and CPU to facilitate the computation-intensive deep-learning task. When the edge cluster has enough resources to process the request, Kubernetes allocates a particular worker node to process the request. As shown in Algorithm 3, *Pod_Allocation(REQtype)* occurs, where the pod is allocated to a particular worker node, and the container is created inside the pod based on the requested service type and is referred to as *ApplicationContainer* (lines 16,17). This means that the container can provide the response to the client-requested data. However, there is a limited distance up to which a particular edge server can provide its service. When the client exceeds a particular distance, the migration of the container is scheduled. The right destination node for container migration is determined using

Algorithms 1 and 2. Once a suitable destination is found, the destination node Internet Protocol (IP) is assigned as *Ed* (line 23). The *ContainerMigration* of the ApplicationContainer is performed by stopping the running container, committing the container that saves the file system modifications inside it as an image, and pushing the modified image into the Docker image storage repository. From the destination node, a particular image is pulled and made to run as a container. Hence, from the destination node E_d, the container can continue providing its service to the client (lines 24–29). In this process, we avoid migrating the container volumes that persist in the container. This is because the real-time service that we provide does not depend on persistent data. The response is immediate for the requested data, which means that, before the client sends the next new data, the previous response reaches the client immediately, and the newly requested data do not depend on the previous data.

Algorithm 3 Service Migration Mechanism

```
1: function MIGRATION(REQ, Distance)
2:
3:    Input:    REQ → Request message from the client
4:               Distance → Distance travelled by the client
5:    Output:  Response → Resultant response to the client
6:    procedure
7:        for eachREQ do
8:            Es ← getCurrentNodeIP()
9:            REQtype ← REQServiceType
10:           REQres ← ResourceRequiredbyREQ
11:           Calculate Rrem
12:       if (REQres ≥ Rrem) then
13:            RequestMigration(REQ)
14:           return ResponsefromEd
15:       else
16:           PodAllocation(REQtype)
17:           ApplicationContainer ← ContainerCreation(REQtype)
18:           if (Distance ≤ Threshold) then
19:               return ResponsefromEs
20:           else
21:                   // Sharing metadata on blockchain
22:                   // Destination node self − evaluation
23:                   Ed ← getDestinationNodeIP()
24:                   ContainerMigration(ApplicationContainer)
25:                       Stop the ApplicationContainer
26:                       Commit it into Image
27:                       Image Transfer to Ed
28:                       Start the ApplicationContainer in Ed
29:               return ResponsefromEd
```

5. Experimental Results

The purpose of the experiments was to evaluate the performance of the proposed algorithms in an ARM-based embedded edge environment that runs deep learning applications using the Compute Unified Device Architecture (CUDA) Toolkit in a container environment.

5.1. Hardware and Software Specifications

In our experiments, edge clusters were built using NVIDIA Jetson development boards using the hardware and software specifications [28] mentioned in Table 2. This was supported by NVIDIA Jetpack and DeepStream Software Development Kits (SDKs), as well as CUDA, CUDA Deep Neural Network library (cuDNN), and TensorRT software libraries. Studies have been conducted on available Jetson boards, and benchmarking has been performed [34] on performance parameters, concluding that Jetson Xavier is more suitable for the deployment of deep learning applications that provide real-time responses to the user. Hence, we developed our edge clusters using NVIDIA Jetson Xavier AGX, which handles the aforementioned real-time, end-to-end, deep learning applications for the connected car environment. In addition, an Intel Wireless-AC 9560 adapter was used as a communication module.

Table 2. Hardware and Software Specifications.

Device	**NVIDIA Jetson Xavier AGX (Jetpack 4.4.1 L4T 32.4.4)**
GPU	512-core Volta GPU with Tensor Cores
CPU	8-core ARM v8.2 64-bit CPU, 8 MB L2 + 4 MB L3
Network Module	Intel AC9560, AGW 200
OS/Kernel	Linux Ubuntu 18.04, Tegra 4.9
Kubernetes	Kubernetes 1.18
Docker	19.03.6
TensorFlow	1.15.0

5.2. Evaluation of Network Performance

The concept of service migration and blockchain connects various edge servers in the edge plane. As a preliminary experiment, to check the configuration of our edge environment, the network bandwidth was measured between the two edge servers configured with the Kubernetes and Docker container environment. iperf3 [35] was the network performance measurement tool used to measure the network performance. One edge server was configured in the server mode and the other in the client mode. Figure 5 shows a normal distribution graph for the variation in network bandwidth, observed for a period of 100 s. It provides the mean and deviation quantities, which infer that the values near to the mean are more frequent in occurrence.

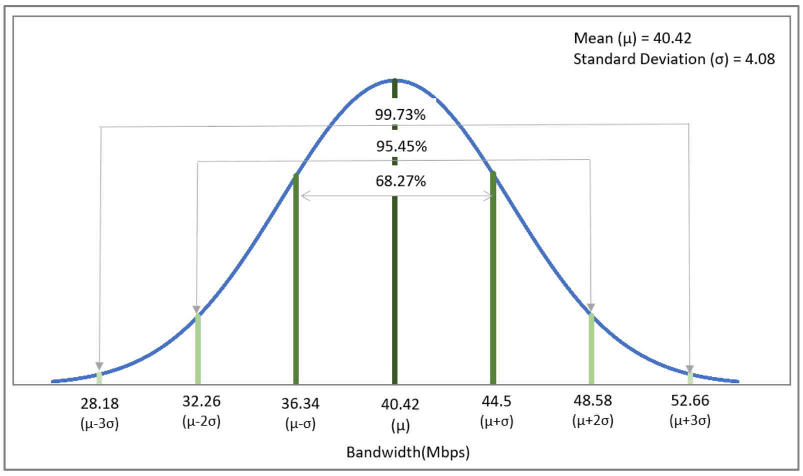


Figure 5. Normal distribution graph for the variation in network bandwidth.

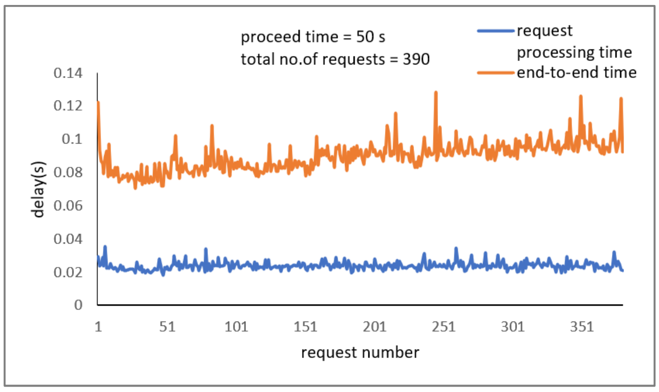
5.3. Experimental Results for the Proposed System Model

First, the performance of the applications on a single-edge server was tested and evaluated using a simulation environment developed in Python. The OCSLab dataset [33] and ImageNet dataset [32] were used to run the driver behavior profiling and image recognition applications, respectively. The applications were made to run in a container environment using the application images. *DockerHub*, a private image storage repository, stored the application images. We assume that the client connected to the nearest edge server to request the information. As a result, the following were observed.

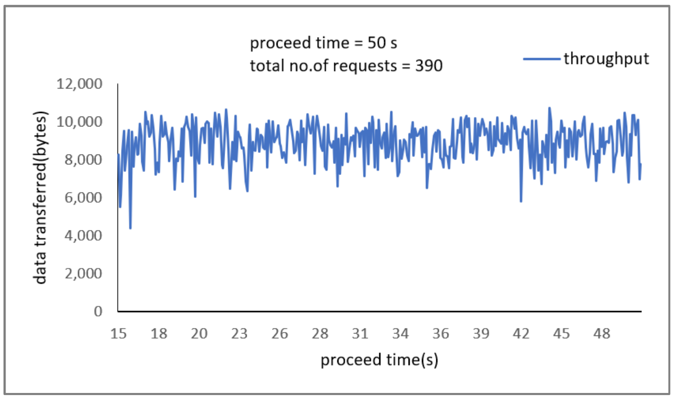
(i) *End-to-end delay* is the request start time from the client until the client receives the response. It includes the transmission delay and the processing delay. The transmission delay is the communication delay between the client and the edge server, and the processing delay is the request in the processing delay by the application in the container environment.

(ii) *Throughput* is the amount of data transferred in bytes with respect to time.

When the client connection is established to the nearest edge server, the request for data and response to the client were performed for a period of 300 seconds (s). For the Driver Behavior Profiling application, the average request processing time was 0.0235 s with a confidence of 99.94 % and end-to-end time of 0.0913 s. The average throughput was 8922.37 bytes per second, containing 3069 requests. Figure 6a,b provide the end-to-end time and throughput performance metrics, respectively, for a period of 50 s containing 390 requests.



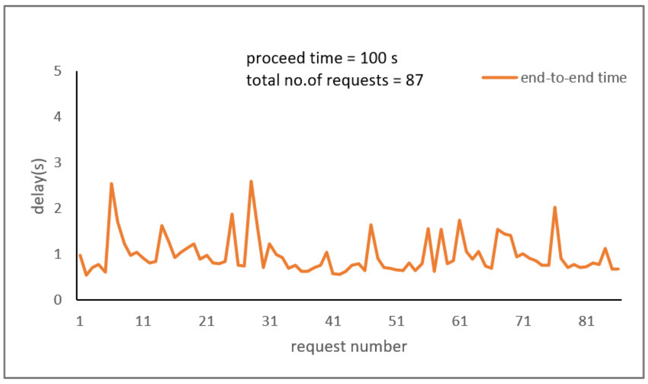
(**a**)



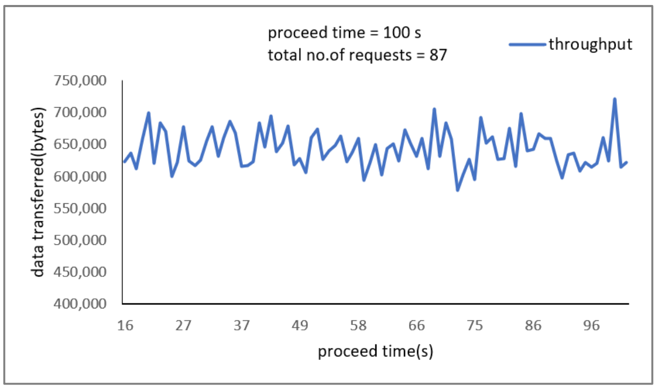
(**b**)

Figure 6. (**a**) End-to-end time graph; (**b**) Throughput graph for Driver Behavior Profiling.

For Image Recognition application, the average end-to-end time was 1.0449 s, along with the object name and the probability value for each object. The average throughput was 638,074.941 bytes per second, containing 285 requests. Figure 7a,b provide end-to-end time and throughput performance metrics, respectively, for a period of 100 s containing 87 requests.



(**a**)



(**b**)

Figure 7. (**a**) End-to-end time graph; (**b**) Throughput graph for Image Recognition.

(iii) *Memory and CPU* are the compute resources that can be requested, allocated and consumed in Kubernetes. These resources are used for the container creation in the Kubernetes edge server environment to run the application and process the request data. Requests and Limits are the mechanisms that the Kubernetes uses to control the Memory and CPU usages. In general, the resource requests and resource limits can be specified for the container by using the *resource: requests* field and *resources: limits* field in the container resource manifest file. In BloSM, memory resources are allocated for the applications. For driver behavior profiling applications, the memory request is allocated as "3,417,969 Kibibyte(Ki)" and memory limit as "3,906,250 Ki". For image recognition applications, the memory request is allocated as "4,882,812 Ki" and memory limit as "5,371,094 Ki". The utilization of CPU by the application containers without allocation was observed to be very minimal.

5.3.1. Scheduling of Service Migration Due to Insufficient Resources

The proposed method for migration due to insufficient resources focuses on the available memory on the edge nodes. When each request reaches the edge server, it first reaches the master node. The master node automatically allocates the job to a suitable worker node. When the *allocatable memory* of all the worker nodes is utilized by many other pods in the cluster, the remaining memory available in the worker nodes may not be sufficient to process the incoming request. If it is insufficient, the service requests remain in the waiting state without processing. Therefore, to avoid the waiting time, the service requests are scheduled for migration to the next nearest edge server. In other words, the requests are load-balanced with nearby available resources. Simulation experiments were performed, and the average request migration time, irrespective of the application, was observed to be 361 ms. This adds to the total delay in processing the request. Figure 8a,b show the end-to-end time with the request number in a request migration scenario for both the applications. The request migration time is indicated with a grey block, followed by the request processing scenario in the next nearest edge cluster, to which the requests migrated. The average end-to-end time was 0.1307 s for driver behavior profiling and 1.3264 s for image recognition, which is increased in comparison with Figures 6a and 7a due to the client connection to the next nearest edge cluster.

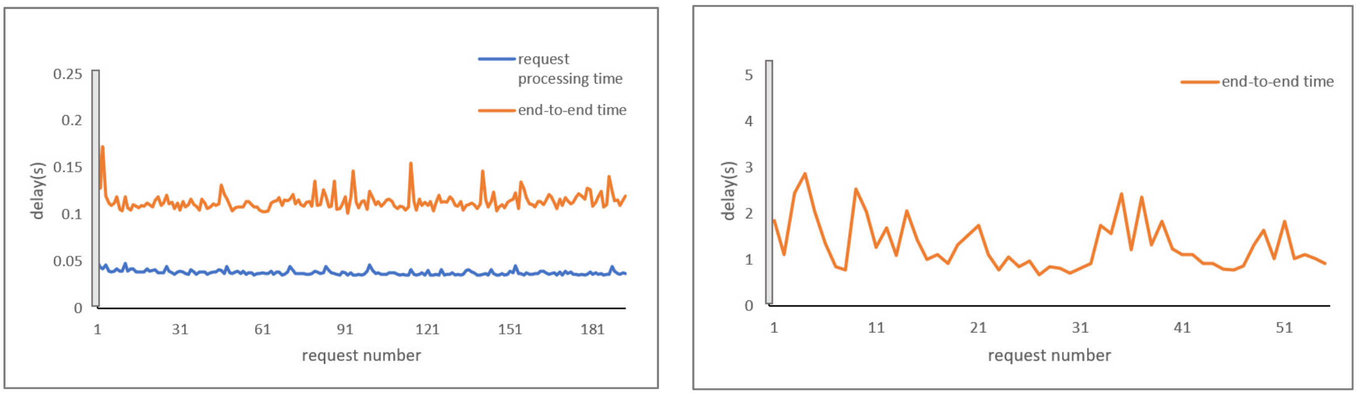
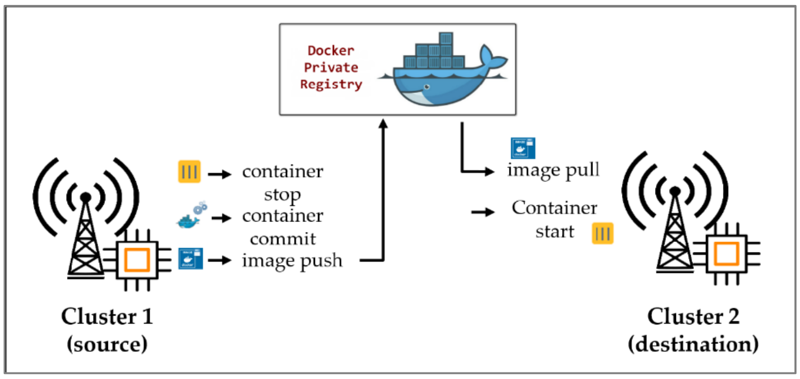


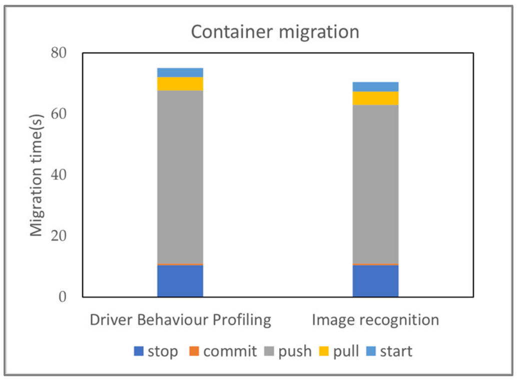
Figure 8. End-to-end time graph in case of request migration. (**a**) Driver behavior profiling; (**b**) Image recognition.

5.3.2. Scheduling of Service Migration Due to Client Mobility

The proposed method of migration due to client movement involves the migration of containers using a modified container image migration method, as shown in Figure 9a. When the client moves away from the distance within which a particular edge server can provide its service, the container created in the source edge cluster is migrated to a suitable destination edge cluster based on Blockchain. Our applications inside the container are GPU-accelerated, using the CUDA Toolkit to run deep learning tasks. In the source edge cluster, the first step is to stop the running container and commit it into an image. The stop time is slightly longer than usual due to the use of GPUs by the container. The image is then pushed to the *DockerHub*, a private storage repository and pulled at the destination. The transfer time depends on the container image size. The docker pushes the modified file systems, and the remaining layers are mounted from the existing application image. The modified layers are downloaded on the destination node, and the remaining layers are mounted from the base and application images that existed on the destination node. In the destination, the container is started using the image to serve the client.



(**a**)

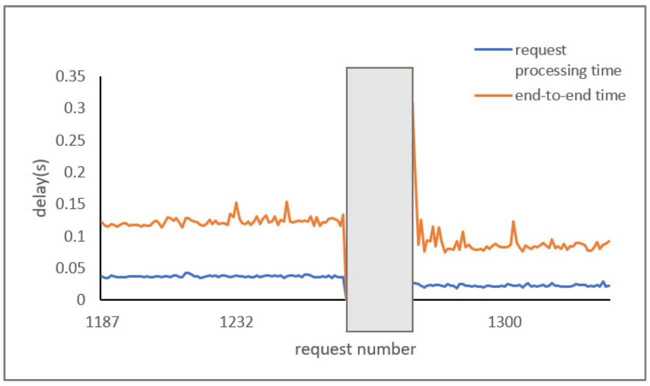


(**b**)

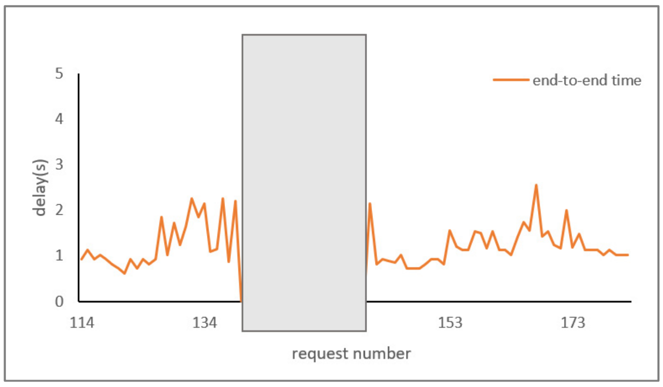
Figure 9. (**a**) Container migration process; (**b**) container migration time consumption.

The total migration time for the 4.4 GB driver behavior profiling application was measured to be 75.11 s and that for the 2.13 GB image recognition application was 70.46 s, as shown in Figure 9b with the split-up time for each step performed during migration. The larger migration times are due to the larger container image sizes and the use of GPUs by the container. By employing the modified image migration method, we ensured that the right version of the application and all the image layers exist in the destination. Missing layers can be downloaded in real time.

Figure 10a,b show the end-to-end time with the request number in case of a container migration scenario for both the applications. The request-processing scenario before and after migration in the source and the destination nodes is shown, respectively. The end-to-end time was observed to be nearly the same due to the client mobility and client connection with its nearest edge server coverage area. The container migration time is indicated with a grey block. The average end-to-end time is 0.1178 s for driver behavior profiling and 1.0863 s for image recognition before migration. The average end-to-end time is 0.1069 s for driver behavior profiling and 0.9513 s for image recognition after migration. The migration time in the middle accounts for the total downtime and adds to the total delay in processing the remaining requests in the destination node.



(**a**)



(**b**)

Figure 10. End-to-end time graph in case of container migration. (**a**) Driver behavior profiling; (**b**) image recognition.

5.3.3. Blockchain-Based Decision-Making Efficiency

To demonstrate the blockchain-based, destination node decision-making procedure, we performed several experiments on a uniformly distributed, sample edge-computing environment. For the simulated environment, we set up 100 blockchain nodes by aggressively manipulating the nodes (meaning that one edge node may simulate several blockchain nodes to create a bigger network). As the applied Blockchain uses PoA consensus, the network has one validator node to approve the metadata transactions. Due to the PoA consensus, the network security and scalability are not affected by the number of nodes; however, more nodes are needed to test the throughput and latency performances for the expected environment. In this scenario, half of the nodes are run on high-performing, server-like computers, and the other half run on embedded boards with limited resources. The experiments were performed looking at four main factors: latency, throughput, storage usage, and bandwidth usage. The overall performance results of the service migration decision-making process with blockchain-based techniques and traditional, central cloud-based techniques are shown in Figure 11. As the evaluation units for these factors are different from each other, we used a common metric cost (lower is better) for all. The basic unit of the cost is 1, which refers to the total cost when the destination node is known, and communication is only between two nodes.

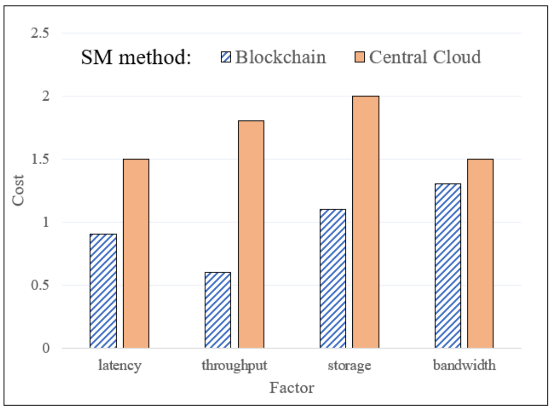


Figure 11. Source to destination communication, (Nodes are uniformly distributed, where 50% have high, and 50% have low resources. Cost value 1 stands for the optimal cost of performing the operation.)

The overall conclusion from the figure is that the blockchain benefits the system with all the factors. The blockchain-based approach decreases the latency to 0.9 and the throughput cost to 0.6 by fairly providing a local blockchain database for all nodes. By contrast, the traditional approach requires centralized computing, where the source node uploads the container to the central cloud, and the destination node migrates it. In this manner, the destination node first sends a request to the cloud for metadata and performs self-evaluation. As a result, the central cloud-based technique increases the latency to 1.5, and the throughput cost to 1.8. Moreover, the proposed technique has advantages in terms of storage utilization. When the migration process is limited to only one-way communication, the total cost is only increased by metadata propagation, which is an almost negligible cost to the network. However, the traditional method requires two-way communication, where the source node is needed to re-run the container from the central cloud when the destination has insufficient resources. Finally, the proposed technique has advantages over centralized computing by eliminating the bandwidth in the central node. Even when the total network bandwidth is evaluated, broadcasting only the metadata is beneficial compared with uploading the container to the central cloud.

6. Discussion

Service migration is one of the major problems in edge computing. In order to address this issue, we provided a solution for two different scenarios, as discussed above. We proposed a novel idea of employing Blockchain to find the right destination for container migration and eliminate further migrations.

In scenario (i), as explained in Section 5.3.1, we propose our own method of migrating service requests based on service density, which helps to load-balance the requests among the edge plane. This migration scenario can be avoided by increasing the resources on the edge clusters to process the requests.

In the case of scenario (ii), as implemented in Section 5.3.2, when comparing the previous studies, Ma et al. [10] performed experiments for the live migration of docker containers by considering the docker's layered storage, which resulted in a total migration time of 3.2 s (2.8 s) and 10.9 s (10.3 s) for Busybox (290 KB) and OpenFace (2.17 GB) containers, respectively, under the network bandwidth of 500 Mbps, and 3.2 s (2.8 s) and 48.9 s (48.1 s) under 5 Mbps. The times mentioned in parentheses denote the service downtime. However, the checkpointing of containers is not adaptable to our environment for the following reasons. First, checkpointing requires kernel support and is kernel-dependent. Second, the checkpointing of GPU containers is not supported at either the software or hardware level in any of the AMD or ARM machines. Bellavista et al. [11] proposed a migration technique that leverages the container characteristics in the MEC/Fog environment by using a docker-compose file for hardware experiments and EdgeCloudSim simulator for simulation environments. When using EdgeCloudSim, the total migration time for reactive handoff is 170 s, and proactive handoff is nearly 43 s for 300 MB (10 K records) data.

Our proposed method, BloSM, carefully studied the configurations of our proposed environment and employed a container migration method that is suitable for GPU-based containers. Integration with Blockchain finds a suitable migration destination, preventing further migrations. The proposed method migrates the container image with the modified file system and eliminates the process of migrating container volumes and their states. In our case, the service downtime is equal to the total migration time for the migration of high-computing GPU containers, which resulted in 75.11 s and 70.46 s. In the case of smaller container sizes, the migration time reduces. For example, with reference to Table 3, a container size of 264 MB takes 20.33 s. By employing the modified image migration method, we ensured that the right version of the application and all the image layers exist in the destination. Missing layers can be downloaded in real time. Higher migration times are due to the larger container sizes. The applications' larger container sizes are due to the number of dependencies, such as OpenCV (built from source), TensorFlow, and several other deep-learning libraries. In general, more installations on the base image lead to large container sizes.

Regarding the cost analysis for the proposed solution, the energy budget range that the Jetson board provides is 10 W~30 W, which is comparatively better than the other server implementations. Jetson Xavier AGX provides 10$\times$ times more energy efficiency than its predecessor, the NVIDIA Jetson TX2 [36]. Additionally, its smaller size is more advantageous for deployment in real-time environments. In terms of financial cost, the Jetson Xavier AGX deployed in our testbed costs around $1000 each, accounting for its having the best performance in terms of AI processing. Here, there is a tradeoff between the cost and the performance.

Considering the vehicular scenario, our proposed environment provides effectivity support by reducing the end-to-end time and providing immediate response to the user, both before and after migration. The container migration time accounts for the service downtime and is slightly higher due to the larger container sizes and GPU usage, as mentioned above. This results in a tradeoff between the container size and migration time. However, the proposed method of migration was designed with a focus on the vehicular environment, where the migration of requests consumes time in milliseconds and container migration time can be made more effective by optimizing the container image sizes that are used to run the application. Additionally, our novel approach, based on blockchain, proved the efficiency of the process in terms of latency, throughput storage, and bandwidth in the edge plane.

Table 3. Comparison with the existing methods.

References	**Proposed Method**	**Container (Size)**	**Migration Time**	**Configurations**	
				Bandwidth	**Environmental Setup**
Ma et al. [10]	Live Migration through Docker's layered storage.	Busybox (290 KB)	3.2 s (2.8 s)	500 Mbps	Desktop server with 2 VM. Client-laptop (CPU)
			3.2 s (2.8 s)	5 Mbps	
		OpenFace (2.17 GB)	10.9 s (10.3 s)	500 Mbps	
			48.9 s (48.1 s)	5 Mbps	
P. Bellavista et al. [11]	reactive handoff	300 MB (10 K records)	170 s	40 Mbps	EdgeCloudSim Simulator
	Proactive handoff		43 s		(CPU)
BloSM	Service RequestMigration	Client Request Message	361 ms	Around 40 Mbps	NVIDIA Jetson Xavier AGX
	Blockchain-based Container Image Migration method	WordPress sample container (264 MB)	20.33 s		(CPU)
		Driver behavior Profiling (4.4 GB)	75.11 s		(CPU & GPU)
		Image Recognition (2.13 GB)	70.46 s		

7. Conclusions

Effective service migration is a challenging topic in edge computing, where network requirements change depending on the environment. This research proposed a new blockchain-based service migration solution named BloSM for edge networks with embedded devices. It addresses the scenario of insufficient resources on the edge nodes by scheduling the migration of requests and client mobility through container migration. In addition, the adaptation of blockchain protocol enables the locality of the real-time migration metadata for all network nodes. It also enables the destination node self-evaluation technique using the metadata, which helps make migration decisions. The evaluations prove the efficient use of edge resources and provide a guaranteed migration for GPU containers with considerable downtime. Blockchain-based metadata-sharing has advantages over the traditional cloud-based method in terms of latency, storage, throughput, and bandwidth.

Author Contributions: Conceptualization, S.K. and K.T.; methodology, S.K. and K.T.; software, S.K. and K.T.; validation, S.K., K.T. and D.-H.K.; formal analysis, D.-H.K.; investigation, S.K. and K.T.; resources, D.-H.K.; data curation, S.K. and K.T.; writing—original draft preparation, S.K. and K.T.; writing—review and editing, D.-H.K.; visualization, S.K. and D.-H.K.; supervision, D.-H.K.; project administration, D.-H.K.; funding acquisition, D.-H.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Institute for Information & Communications Technology Promotion (IITP) grant, funded by the Korean government (MSIT) (No.2019-0-00064, Intelligent Mobile Edge Cloud Solution for Connected Car), and in part by a National Research Foundation of Korea (NRF) grant, funded by the Korean government (MSIT) (No. NRF-2021R1F1A1050750).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. *Decent. Bus. Rev.* **2008**, 21260.
2. Yang, R.; Yu, F.R.; Si, P.; Yang, Z.; Zhang, Y. Integrated blockchain and edge computing systems: A survey, some research issues and challenges. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 1508–1532. [CrossRef]
3. Guo, S.; Hu, X.; Guo, S.; Qiu, X.; Qi, F. Blockchain meets edge computing: A distributed and trusted authentication system. *IEEE Trans. Ind. Inform.* **2019**, *16*, 1972–1983. [CrossRef]
4. Satyanarayanan, M. The emergence of edge computing. *Computer* **2017**, *50*, 30–39. [CrossRef]
5. Abbas, N.; Zhang, Y.; Taherkordi, A.; Skeie, T. Mobile edge computing: A survey. *IEEE Internet Things J.* **2017**, *5*, 450–465. [CrossRef]
6. Wang, S.; Zhao, Y.; Xu, J.; Yuan, J.; Hsu, C.H. Edge server placement in mobile edge computing. *J. Parallel Distrib. Comput.* **2019**, *127*, 160–168. [CrossRef]
7. Ahmed, E.; Rehmani, M.H. Mobile edge computing: Opportunities, solutions, and challenges. *Future Gener. Comput. Syst.* **2017**, *70*, 59–63. [CrossRef]
8. Machen, A.; Wang, S.; Leung, K.K.; Ko, B.J.; Salonidis, T. Live service migration in mobile edge clouds. *IEEE Wirel. Commun.* **2017**, *25*, 140–147. [CrossRef]
9. Wang, S.; Xu, J.; Zhang, N.; Liu, Y. A survey on service migration in mobile edge computing. *IEEE Access* **2018**, *6*, 23511–23528. [CrossRef]
10. Ma, L.; Yi, S.; Li, Q. Efficient service handoff across edge servers via docker container migration. In Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC '17), New York, NY, USA, 12–14 October 2017; pp. 1–13. [CrossRef]
11. Bellavista, P.; Corradi, A.; Foschini, L.; Scotece, D. Differentiated service/data migration for edge services leveraging container characteristics. *IEEE Access* **2019**, *7*, 139746–139758. [CrossRef]
12. Das, A.; Patterson, S.; Wittie, M. Edgebench: Benchmarking edge computing platforms. In Proceedings of the 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Zurich, Switzerland, 17–20 December 2018; pp. 175–180. [CrossRef]
13. Zhou, Z.; Chen, X.; Li, E.; Zeng, L.; Luo, K.; Zhang, J. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. In Proceedings of the IEEE, Banda Aceh, Indonesia, 22–24 August 2019; pp. 1738–1762. [CrossRef]
14. Huh, J.H.; Seo, Y.S. Understanding edge computing: Engineering evolution with artificial intelligence. *IEEE Access* **2019**, *7*, 164229–164245. [CrossRef]
15. Wang, S.; Urgaonkar, R.; Zafer, M.; He, T.; Chan, K.; Leung, K.K. Dynamic service migration in mobile edge computing based on Markov decision process. *IEEE ACM Trans. Netw.* **2019**, *27*, 1272–1288. [CrossRef]
16. Zhang, W.; Hu, Y.; Zhang, Y.; Raychaudhuri, D. Segue: Quality of service aware edge cloud service migration. In Proceedings of the 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Luxembourg, Luxembourg, 12–15 December 2016; pp. 344–351. [CrossRef]
17. Lee, J.; Kim, J.; Tae, Y.; Pack, S. QoS-aware service migration in edge cloud networks. In Proceedings of the 2018 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia), JeJu, Korea, 24–26 June 2018; pp. 206–212. [CrossRef]
18. Yin, L.; Li, P.; Luo, J. Smart contract service migration mechanism based on container in edge computing. *J. Parallel Distrib. Comput.* **2021**, *152*, 157–166. [CrossRef]
19. Zhang, C.; Zheng, Z. Task migration for mobile edge computing using deep reinforcement learning. *Future Gener. Comput. Syst.* **2019**, *96*, 111–118. [CrossRef]
20. Zheng, Z.; Xie, S.; Dai, H.; Chen, X.; Wang, H. An overview of blockchain technology: Architecture, consensus, and future trends. In Proceedings of the 2017 IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, USA, 25–30 June 2017; pp. 557–564. [CrossRef]
21. Xiong, Z.; Zhang, Y.; Niyato, D.; Wang, P.; Han, Z. When mobile Blockchain meets edge computing. *IEEE Commun. Mag.* **2018**, *56*, 33–39. [CrossRef]
22. Zhang, X.; Wu, W.; Yang, S.; Wang, X. Falcon: A Blockchain-Based Edge Service Migration Framework in MEC. *Mob. Inf. Syst.* **2020**, *2020*, 8820507. [CrossRef]
23. Aujla, G.S.; Singh, A.; Singh, M.; Sharma, S.; Kumar, N.; Choo, K.K.R. BloCkEd: Blockchain-based secure data processing framework in edge envisioned V2X environment. *IEEE Trans. Veh. Technol.* **2020**, *69*, 5850–5863. [CrossRef]
24. Pahl, C.; El Ioini, N. Blockchain Based Service Continuity in Mobile Edge Computing. In Proceedings of the 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS), Granada, Spain, 22–25 October 2019; pp. 136–141. [CrossRef]
25. Rad, B.B.; Bhatti, H.J.; Ahmadi, M. An introduction to docker and analysis of its performance. *Int. J. Comput. Sci. Netw. Secur.* **2017**, *17*, 228.
26. Ismail, B.I.; Goortani, E.M.; Ab Karim, M.B.; Tat, W.M.; Setapa, S.; Luke, J.Y.; Hoe, O.H. Evaluation of docker as edge computing platform. In Proceedings of the 2015 IEEE Conference on Open Systems (ICOS), Melaka, Malaysia, 24–26 August 2015; pp. 130–135. [CrossRef]
27. Build and Run Docker Containers Leveraging NVIDIA GPUs. Available online: https://github.com/NVIDIA/nvidia-docker (accessed on 12 November 2021).

28. Kim, J.; Ullah, S.; Kim, D.-H. GPU-based embedded edge server configuration and offloading for a neural network service. *J. Supercomput.* **2021**, *77*, 8593–8621. [CrossRef]
29. Burns, B.; Beda, J.; Hightower, K. *Kubernetes: Up and Running: Dive into the Future of Infrastructure*, 2nd ed.; O'Reilly: Sebastopol, CA, USA, 2019.
30. Bernstein, D. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Comput.* **2014**, *1*, 81–84. [CrossRef]
31. Ullah, S.; Kim, D.-H. Lightweight Driver Behavior Identification Model with Sparse Learning on In-Vehicle CAN-BUS Sensor Data. *Sensors* **2020**, *20*, 5030. [CrossRef] [PubMed]
32. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 4510–4520.
33. Kwak, B.I.; Woo, J.; Kim, H.K. Know your master: Driver profiling-based anti-theft method. In Proceedings of the 2016 14th Annual Conference on Privacy, Security and Trust (PST), Auckland, New Zealand, 12–14 December 2016; pp. 211–218. [CrossRef]
34. Ullah, S.; Kim, D.-H. Benchmarking Jetson platform for 3D point-cloud and hyper-spectral image classification. In Proceedings of the 2020 IEEE International Conference on Big Data and Smart Computing (BigComp), Busan, Korea, 19–22 February 2020; pp. 477–482. [CrossRef]
35. iPerf3 and iPerf2 User Documentation—iPerf. Available online: https://iperf.fr/iperf-doc.php (accessed on 12 November 2021).
36. Jetson AGX Xavier Developer Kit. Available online: https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit (accessed on 30 December 2021).