Article

Microservices-Based Resource Provisioning for Multi-User Cloud VR in Edge Networks

Ho-Jin Choi 1,2, Nobuyoshi Komuro 3 and Won-Suk Kim 1,2,*

1 The Department of Computer Engineering, Pusan National University, Busan 43241, Republic of Korea; tiger981228@pusan.ac.kr
2 Center for Artificial Intelligence Research, Pusan National University, Busan 43241, Republic of Korea
3 Institute of Management and Information Technologies, Chiba University, Chiba 263-8522, Japan; kmr@faculty.chiba-u.jp
* Correspondence: wonsukkim@pusan.ac.kr

Abstract: Cloud virtual reality (VR) is attracting attention in terms of its lightweight head-mounted display (HMD), providing telepresence and mobility. However, it is still in the research stages due to motion-to-photon (MTP) latency, the need for high-speed network infrastructure, and large-scale traffic processing problems. These problems are expected to be partially solved through edge computing, but the limited computing resource capacity of the infrastructure presents new challenges. In particular, in order to efficiently provide multi-user content such as remote meetings on edge devices, resource provisioning is needed that considers the application's traffic patterns and computing resource requirements at the same time. In this study, we present a microservice architecture (MSA)-based application to provide multi-user cloud VR in edge computing and propose a scheme for planning an efficient service deployment considering the characteristics of each service. The proposed scheme not only guarantees the MTP latency threshold for all users but also aims to reduce networking and computing resource waste. The proposed scheme was evaluated by simulating various scenarios, and the results were compared to several studies. It was confirmed that the proposed scheme represents better performance metrics than the comparison schemes in most cases from the perspectives of networking, computing, and MTP latency.

Keywords: edge computing; cloud computing; cloud VR; microservice architecture; network management; cloud-native; edge-native; metaverse

1. Introduction

Virtual reality (VR) is a technology that provides a visually realistic representation of a virtual environment. It is used for immersive content, training simulations, and remote control. Cloud VR is a technology that streams rendered views from the cloud server to the head-mounted display (HMD) for playback in order to overcome limitations such as performance, battery life, portability, and mobility of the HMD. In cloud VR, motion-to-photon (MTP) latency, which is the delay between the user’s motion input and the result being transmitted back to the HMD for playback, becomes a critical issue [1–4]. If the MTP latency exceeds 20 ms, VR sickness can occur, negatively impacting the user experience [5,6].

The largest portion of this latency is due to the physical distance between the HMD and the remote server, as well as the huge amount of traffic that occurs during streaming. The physical distance between remote servers increases latency, as the signal has a longer path to travel, which is a major issue in VR environments where real-time responsiveness is critical. The huge amount of traffic that occurs during streaming causes network bottlenecks, which reduce the efficiency of data processing and transmission, causing delays. These bottlenecks can affect the entire network system, potentially degrading the performance of
other services using the same network. Therefore, such latency is a key consideration in cloud VR resource provisioning [7–9].

However, cloud computing encounters various issues like high latencies, expensive migration costs, lack of location awareness and mobility support, and limited adaptability in terms of communication types [10]. Therefore, edge computing can be considered the most effective solution to solving the MTP latency issue in cloud VR [11–13]. By utilizing edge computing, the rendering function of cloud VR applications can be processed at the edges of the network, which reduces network delays and thus reduces MTP latency. However, compared to data center hosts, edge devices have relatively low computing resource capacities and high device heterogeneity, and the location of each device is dependent on the network topology [14–16]. Therefore, cloud VR servers should be deployed and managed by simultaneously considering the computing resource capacity of edge devices in the network and the network topology. Moreover, for multi-user cloud VR applications such as the metaverse or teleconferencing, the computing load requirements of a single edge server increase, making it difficult to discover suitable edge devices for deployment. This means that it is difficult to satisfy the MTP latency thresholds of distributed multiple users simultaneously [17–19].

1.1. Motivations

The playback process for single-user cloud VR consists of several steps, as shown in Figure 1. The steps are as follows: (1) motion input signal processing inside the HMD, (2) input signal transmission to content server, (3) engine processing on the server, (4) rendering and encoding for video streaming, (5) field of view (FoV) streaming, (6) decoding, and (7) playback of the video stream. In steps (2) and (5), wired and wireless communication is performed, and in steps (3) and (4), the computing resources of the edge node where the server operates are used.

![Figure 1. Entire process from motion input to playback in cloud VR.](image-url)

In order to provide multi-user cloud VR in an edge computing environment, it is essential to respond to the following challenging issues.

First, in a multi-user environment, some steps change. Steps (1) and (2) are performed separately on each user’s HMD, while step (3) must be performed at a single point because the same environmental information must be synchronized for all users. For (4), the FoV rendering must be performed separately for each user, since their current position and viewing direction are both different. This result is played back through (5)–(7) on each
user’s HMD. In such cases, as rendering must be performed individually for each user simultaneously connected to a single server, it requires computing resources that are specific to each user’s rendering needs. Thus, as the number of simultaneously connected users increases, the rendering load increases linearly, making it difficult for a single edge server to handle every processing step.

Second, the MTP latency of an individual user is determined by the network distance between each user and the server that receives their motion input and performs the rendering. In order to provide MTP latency below the threshold for an individual user, the rendering processing server should be deployed as close to that user as possible. The location of the server on the edge network has a critical impact on the user experience of cloud VR. However, the edge network’s limited computing and networking resources, along with the decreasing capacity of computing resources in nodes closer to the network edge where users are located, can make it a difficult problem to solve [20,21]. To address these issues, there are studies that try to solve them by dividing functions based on microservices. However, the proposed approaches are not suitable for multi-user environments [22,23].

Third, in a traditional monolithic application structure, processes (3) and (4) are performed on a single server. In a single-user environment, the resource requirements of cloud VR are low enough that it can be deployed at the edge, even if all processes are performed on a single server. However, in a multi-user environment, the resource requirements of cloud VR increase, so it cannot be deployed at the edge where resources are limited and may be involved in the core network. Therefore, to avoid involving services up to the core network, it is necessary to minimize the resource footprint within the edge network by balancing resources according to the service’s resource requirements.

1.2. Contributions

In this study, to respond to the challenging issues of multi-user cloud VR, we restructure the cloud VR application to follow MSA and to provision microservices to ensure MTP latency thresholds for all users while efficiently using networking and computing resources. A conceptual diagram of the MSA-based cloud VR applications at the network edge is shown in Figure 2. In such an environment, the proposed scheme has the following features.

- The rendering load of the edge server inevitably increases linearly with the number of users. Continuing to respond with a single server creates a scalability issue, so we aim to solve the problem by applying MSA. The adoption of a microservices architecture yields several benefits, that are crucial for optimizing system performance and enhancing operational efficiency [24]. Tasks that require single-point processing are configured with a single microservice called an engine service. The engine service mainly aggregates multiple user inputs (actions and motions) based on timestamps and computes the results to be reflected in the next frame of the content. Tasks such as user FoV rendering are configured with a render service for each user.

- As the number of users increases, resources in the edge network might become insufficient due to the high computing resource demand of the render service, which may be placed far from the user and may not satisfy the MTP for all users. To address this, we propose a strategy to separate the microservices responsible for the rendering process into render and motion services. The render service performs most of the rendering tasks, while the motion service provides fast response times by performing only simple processing such as motion input-based video cropping on the rendered video stream. With this separation, the motion service has a relatively low computing load and is more likely to be deployed close to users, which allows for motion input processing to be performed quickly.

- The cloud VR application proposed in this study is assumed to be composed of microservices such as engine services, render services, and motion services. The engine service can be deployed anywhere in the network due to low CPU and GPU loads and low network bandwidth requirements. The render service typically has a high CPU and GPU load and can have high network bandwidth requirements for
streaming because it carries additional information in addition to the video stream to support the motion service's work alone. The motion service has a relatively low computational load, and the resulting FoV stream also has a relatively low network bandwidth load. The proposed scheme develops a placement plan that reflects these microservice-specific workload characteristics. We propose a strategy that allows as many users as possible to operate within the edge while balancing resources and considering the service's characteristics as well.

Figure 2. Conceptual diagram of the MSA-based cloud VR applications at the network edge.

2. Related Works

In our previous work [25], we proposed a resource provisioning scheme that comprehensively considers content types, client usage patterns, and network metrics to satisfy the user experience of cloud VR applications. However, in order to provide multi-user cloud VR, an MSA configuration is inevitable, which means that the existing single-server-based resource provisioning scheme cannot consider the characteristics of each service. This study extends the concept proposed in our previous study to multi-user environments to establish a deployment plan that considers the characteristics of each microservice and satisfies the MTP latency requirements of all users.

Alencar et al. [22] Table 1 divided VR into modules with user information (i.e., device, buffer, playback quality), and each module was assigned to handle VR content for microservices. When deploying the system, only latency was considered to satisfy the quality of experience (QoE) of VR streaming, while the computational load of each module was not taken into account. Fog4VR determines the optimal fog node to allocate the VR microservices based on delay, migration time, and resource utilization rate. Rigazzi et al. [23] proposes a novel solution for decomposing and distributing the end-to-end 360 video streaming service across three computing tiers, namely cloud, edge, and constrained fog, in order of proximity to the end user client. These studies involve dividing applications into microservices and deploying them at the edge. However, while deployments are made to satisfy latency requirements, there is often a lack of consideration for the computing resource demands of the services. This presents a limitation, as there is the potential to fail to meet the latency requirements in multi-user environments or when computing resources are lacking.

Du et al. [26] proposed collaborative virtual reality, a cloud-based multi-user VR headset system that enables real-time communication among users in an interactive VR environment. Hou et al. [27] proposed a wireless VR system based on edge and cloud computing to overcome the limitations of HMDs, and they proposed appropriate countermeasures for bitrate and latency requirements. Zhao et al. [2] built a VR testbed utilizing
cloud servers, commercial VR headsets, and general Wi-Fi access points (APs) and analyzed cloud VR application traffic and QoS characteristics to improve content performance and user experience. These studies assume that cloud VR applications are provided to a single user or a small number of users, and there is a limitation, in that there is a lack of consideration for the computing resource requirements of the service.

Table 1. Comparison of related works.

<table>
<thead>
<tr>
<th>Related Works</th>
<th>Subject</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alencar et al. [22] Rigazzi et al. [23]</td>
<td>Microservice-based VR system</td>
<td>Insufficient consideration of the computing resource requirements</td>
</tr>
<tr>
<td>Zhao et al. [2] Du et al. [26] Hou et al. [27]</td>
<td>Cloud-based VR headset system</td>
<td>A single user or a small number of users</td>
</tr>
<tr>
<td>Hu et al. [28]</td>
<td>Edge-based face recognition system</td>
<td>Only focuses on applications</td>
</tr>
<tr>
<td>Xu et al. [29] Alam et al. [30]</td>
<td>Fog server deployment</td>
<td>Insufficient consideration of edge server deployment location</td>
</tr>
<tr>
<td>Velasquez et al. [31] Taneja et al. [32] Zhang et al. [33] Wang et al. [34] Li et al. [35]</td>
<td>Service placement architecture for IoT environments</td>
<td>Insufficient consideration of the computing resource requirements</td>
</tr>
<tr>
<td>Wang et al. [36] Ma et al. [37]</td>
<td>Microservice-oriented service placement mechanism</td>
<td>Insufficient consideration of the latency requirements</td>
</tr>
<tr>
<td>Current work</td>
<td>Microservice-based cloud VR microservice deployment</td>
<td>Consideration of the latency requirements, the computing resources, and the multiple users</td>
</tr>
</tbody>
</table>

Hu et al. [28] preprocessed images collected from user devices at fog nodes adjacent to the user to implement a face recognition system and then deliver them to the cloud. This distributes the computing load requirements of the application to all fog nodes, and since only the processed results are sent to the remote server, the required network bandwidth is also reduced. Xu et al. [29] proposed a framework for analyzing service request packets and automatically distributing docker containers for specific applications to the edge network. A scheme for effectively managing computing and networking resources in the network was also proposed by automatically removing specific containers when they are no longer needed. Alam et al. [30] proposed a modular and scalable architecture based on lightweight virtualization of the Docker container and edge computing to efficiently orchestrate microservices in IoT environments. This, combined with Docker orchestration, enabled management simplification and distributed deployment, which provided a dynamic system that could distribute fault tolerance and system availability across the application layer. These studies focused on the management efficiency of computing and networking resources within edge networks, but it did not take into account the demands on multi-user services.

Velasquez et al. [31] proposed a service placement architecture for IoT environments. The architecture mainly focuses on a module called a service orchestrator, which includes a model and implementation details for service placement tasks and considers network resources when placing services to minimize the number of hops. However, this approach does not consider the management of computing resources, which may lead to inefficient management of the remaining resources. Taneja et al. [32] first sorted the edge nodes and
servers to be deployed in order of their current computing resource capacity and least demand, respectively. After checking whether the servers can be deployed to the nodes in the sorted order, a plan is made. In other words, services with lower computational resource requirements are prioritized for deployment to nodes with fewer remaining computational resources. However, while this scheme has the advantage of reducing server placement overhead, it does not take latency into account when planning deployments, which can be problematic if there are deadline-driven tasks.

Zhang et al. [33] designed an edge computing-based smart farming system and proposed a service offloading-oriented server placement scheme to optimize data transmission delay and load balance among edge servers. Wang et al. [36] proposed a microservice-oriented service placement mechanism for the internet of vehicles based on mobile edge computing (MEC). It performed integer linear programming optimization to reduce the response latency and computing resource usage. Ma et al. [37] proposed an iterative caching update scheme that considers cooperation among edge nodes and performs cooperative service caching and workload scheduling in MEC. However, it has the limitation that it is difficult to satisfy the latency requirements of cloud VR, which requires real-time interactions.

Wang et al. [34] and Li et al. [35] conducted research to reduce the total latency. They proposed that if it is determined that a particular task cannot be processed within the local system in a timely manner, the computing latency can be reduced by utilizing the sufficient available computing resources of the edge nodes, even at the expense of transmission latency. This is a good way to reduce the total latency through local offloading, but it is limited in its applicability to multi-user cloud VR applications where servers are essential.

3. Main Algorithm

This study proposes an edge computing resource provisioning scheme to reduce user MTP latency in multi-user cloud VR. The proposed scheme aims to provide minimum MTP latency to each user by deploying cloud VR applications in three types of microservices to meet the multi-user environment. Moreover, resource management is also performed to reduce the waste of computing resources in the edge network so that the edge network can accommodate more applications.

3.1. Organizing Microservices for Multi-User Cloud VR Applications

The multi-user cloud VR application is structured based on MSA, as shown in Figure 3. The cloud VR application consists of one engine service and user-specific render and motion services. As previously mentioned, in a multi-user environment, the rendering process is performed individually for each user; hence, user-specific render services are established for scalability. These services carry a significant computing load, so they may not be deployed close to the users, which adversely affects the MTP latency. Therefore, a lightweight motion service is additionally configured on the front side of the render service to quickly process user motion inputs.

The engine service is a core service of the application that implements parts requiring single-point processing. It synchronizes various inputs from multiple users, such as interactions, movements, and motions, to compute content logic. If the content is based on a physics engine, it performs engine computations frame-by-frame based on user inputs, and if it is based on other logic, it executes commands according to that logic. User inputs need to be synchronized, considering logic deciding who pressed a button first; thus, the most suitable form for handling this is a single service. Additionally, validation of user inputs, storage, and utilization of content data are also effectively performed in this single service. The engine service transmits the results of frame-by-frame logic computations to each render service.

The render and motion services are responsible for rendering and streaming the video currently being viewed by a user. For example, if an application is used simultaneously by three users, there will be three render services and three motion services. The render service
performs most of the rendering processes for that user based on the logic computation results of the engine service and the user’s motion inputs.

![Diagram](image)

**Figure 3.** Configuration and operation of MSA in multi-user cloud VR applications.

MTP latency will cause a motion feedback lag that generates the mismatch of motion signals between the vestibular and visual channels [38]. Motion processing can be performed at the edge in order to anticipate the FoV requested by the user. This component can loosen the VR latency requirement where the network cannot support it [39]. In other words, the MTP latency is determined by how quickly the remote server can respond and reflect the changed view based on the motion input. The motion service quickly responds to the user by performing simple operations like video cropping based on the video stream and additional information received from the render service and motion inputs from the user. As this service has a relatively low computing load, it can be deployed near the respective user, maintaining a low level of MTP latency.

The operation of render and motion services can greatly vary depending on the type of content or implementation scheme. For instance, in the case of panoramic video-based content, the render service streams the full view (FV), which is the entire panoramic image centered around the user, and the motion service then crops only the FoV portion based on the current frame’s motion input to deliver to the user. In the case of stereoscopic rendering, the render service performs binocular rendering for a slightly wider range than the current heading’s FoV, also transmitting key object depth information—the distance from the user. The motion service then uses this depth information to fine-tune the position of key objects according to the motion rotation, creating and providing a temporal FoV image. In this case, a normal screen is displayed by the render service a few frames later, as long as the user does not rotate the HMD too dynamically. There can be various other implementations, but it may not be appropriate to cover all of these examples.

While separating the render and motion services brings benefits in terms of MTP latency, the additional overhead associated with transmitting the resulting stream of the render service is inevitable. In other words, the render service must provide additional information to allow the motion service to perform motion processing independently. Examples include FV or depth information. Therefore, it is most ideal for the render and motion services to operate on the same edge device. In this case, networking load can be reduced or motion processing can be directly performed by the render service, thus reducing the overhead between the separated services.

User inputs can be classified into motions, gestures, and interactions. Gestures and interactions are immediately processed and shared by the engine service. Motion is pro-
cessed in all three places: the motion service provides an immediate and temporary FoV configuration response, the render service performs accurate rendering for the current FoV, and the engine service uses it for logic computation or avatar movement to be shared with other users. When a user generates a motion input by turning their head, they first receive a temporarily computed screen from the motion service for playback, followed by the accurately processed image from the render service a few frames later.

Each service requires specific computing resources, such as CPU, memory, storage, and GPU, depending on its operational characteristics. Services primarily run on computing devices in the edge network as containers, and these devices are referred to as edge nodes. Edge nodes can be network devices like Wi-Fi APs, L2-L7 switches, mobile base stations, and road side units (RSUs), or they can be separately connected computing devices such as cache servers or general-purpose servers.

3.2. Microservice Deployment Algorithm for User MTP Latency and Computer Resource Balancing

The goal of the proposed microservice deployment algorithm is to meet the MTP latency threshold for all users of a multi-user cloud VR application while reducing the waste of networking and computing resources within the edge network. Deployment is not conducted for each application individually; rather, services are deployed sequentially across all applications. If deployment is performed application-by-application, once an application’s deployment is complete, the computing resources that have already been assigned may prevent services from being optimally positioned. This could result in services being placed at edge locations far from the user, potentially increasing MTP latency or traffic for some users.

Reducing the waste of computing resources requires a balanced use of resources which equalizing the utilization of each type of computational resource per node. If a type of service heavily utilizes CPU resources, concentrating the same type of service on one node could exhaust its CPU resources, rendering it unable to accommodate other services, despite having ample reserves of other resources. This is an example of unbalanced usage of computing resources.

The most dramatic impact on MTP latency can be attributed to the deployment location of the motion service. Thus, the first priority in the proposed algorithm is to deploy the user-specific motion services of all applications to the edge nodes closest to each user. This is prioritized over other provisioning issues such as the deployment of engine and render services and the balanced use of computational resources.

Next, the user-specific render services for all applications are deployed. These services have high computing resource requirements and generate substantial streaming traffic; therefore, they must be carefully deployed by considering these two factors together. For example, a deployment that minimizes traffic can disrupt the balance of computing resources, and a deployment that prioritizes balance can lead to network congestion. To address these challenges, expected traffic and computing resource usage rates are normalized, and the deployment location is determined based on set weights.

Lastly, the engine service for each application is deployed. Although the engine service plays an important role in providing the application, its required resources are relatively lower or differ in type compared to the render services. The resulting traffic from this service is very low compared to other services, so it does not have a significant impact on the network. Also, no matter which edge node it is deployed on, the latency due to network distance does not significantly impact the user experience, so this service is deployed primarily for balanced resource usage.

Algorithm 1 illustrates the scheme for deploying microservices in a multi-user cloud VR within an edge network. This algorithm considers the existence of various types of cloud VR applications within the network. It is clear that the deployment plan for motion services across all applications are established first, followed by the render services and then the engine services. Notably, only when establishing the deployment plan for render services are the weights in the balance between traffic and computing resources considered.
Algorithm 1: Microservice deployment on the edge network.

1. **Input:**
2. \( A \leftarrow \) the list of cloud VR applications
3. \( N \leftarrow \) the list of edge nodes
4. \( w \leftarrow \) the weight value between expected traffic and computing resource balance
5. \( P \leftarrow \) the map for deployment plannings

6. \( P \).append\( ( \text{CreateMotionDeploymentPlan}(A, N) ) \)
7. \( P \).append\( ( \text{CreateRenderDeploymentPlan}(A, N, w) ) \)
8. \( P \).append\( ( \text{CreateEngineDeploymentPlan}(A, N) ) \)

10. **Output:**
11. return \( P \)

3.2.1. Motion Deployment Plan Algorithm

Algorithm 2 presents the detailed algorithm for establishing the deployment plan for motion services. This algorithm is divided into two main stages.

The first stage calculates the deployment metric for all motion services, and the second stage determines the target node for deployment based on the metric priority.

In the first stage, motion services for each application are extracted, and a deployment metric is computed for each individual service. Here, it is assumed that the engine, render, and motion services are already configured as containers for each application. Lines 10 and 11 select the edge node \( n_c \) connected to the client mapped to the specific service.

Motion services are primarily deployed on node \( n_c \) connected to the client, meaning there is no impact on edge network traffic. Even if deployed on nearby nodes, the impact on network traffic is minimal. Therefore, the deployment metric considers only the balance of computing resources. From the perspective of node computing resources, if any resource is exhausted by deployed services, the remaining resources are nearly unusable. Hence, general bin packing optimization cannot be used, and instead, the vector bin packing (VBP), which also considers resource balance, is utilized. In simple terms, in VBP, the lower the inner angle between the vectorized bins and items, the more the resource distribution matches. Placing items that match the resource distribution in a bin allows for the efficient use of all resources in that bin. Thus, the lower the inner angle between the resource vectors of a node and a service, the higher the deployment priority.

Lines 12–14 vectorize the computing resources of node \( n_c \) and service \( s_m \). Here, the resources of node \( n_c \) are considered post-deployment planning at the current moment. When planning for motion services deployment, the node’s resource capacity is the same as initially. Resource types can include CPU, GPU, memory, storage, etc., and network administrators can choose one or more resources for consideration. Resource loads can be abstracted per resource type, such as cycle time for CPU, bytes for memory, etc. The proposed algorithm calculates the inner angle of the two vectors and then adds them to the priority queue \( q \). The priority queue \( q \) consists of 3-tuple elements (metric, service, node), where the lower the metric, the higher the priority. Thus, Line 15 adds a tuple with the vector angle as the metric \( s_m \) as the service to be deployed and \( n_c \) as the target node to the queue.

The deployment plan entries from the priority queue with the lowest metrics are then processed in order. Lines 22–24 verify that the resource requirements of the entry service are checked to determine whether they can be accommodated by the current processing resources of the entry node. If possible, the corresponding entry service is added to the plan map under the entry node key, and the node’s processing resources are reduced by the service’s resource requirements. The plan map has the following structure:

\[ p : key = n_0, value = [s_0, s_4, s_8, ...] \]
Algorithm 2: CreateMotionDeploymentPlan

1: **Input:**
2:   A ← the list of cloud VR applications
3:   N ← the list of edge nodes
4:
5:   p ← the map for motion services deployment planning, initialized with each node as a key and an empty list of services as the corresponding value
6:   q ← the priority queue for 3-tuple (metric, service, node)
7:   for all a ∈ A do
8:     S_a^motion ← the list of motion services of application a
9:   for all s_m ∈ S_a^motion do
10:       c ← s_m.client // the corresponding client
11:       n_c ← the edge node to which the client c is directly connected, n_c ∈ N
12:       r_n ← [n_c.cpu, n_c.gpu, n_c.mem, ...]^{planned}
13:       r_e ← [s_m.cpu, s_m.gpu, s_m.mem, ...]^{required}
14:       θ ← arccos(((r_n^cpu · r_e^cpu) / \|r_n\| \|r_e\|))
15:       q.enqueue((θ, n_c, e))
16:   end for
17:   end for
18:   while !q.empty() do
19:       e ← q.dequeue()
20:       r_n ← [e.n.cpu, e.n.gpu, e.n.mem, ...]^{processing}
21:       r_e ← [e.s.cpu, e.s.gpu, e.s.mem, ...]^{required}
22:       if x_i < y_i, \forall i \in r_n, y_i \in r_e then
23:           p.get(e.n).add(e.s)
24:           e.n.r^{processing} ← e.s.r^{required}
25:       else
26:           n_r ← the review node, which is the next selected from the sorted list based on expected traffic to n_e
27:           if n_r is null then return -1
28:           r_r ← [n_r.cpu, n_r.gpu, n_r.mem, ...]^{processing}
29:           θ ← arccos(((r_r^cpu · r_e^cpu) / \|r_r\| \|r_e\|))
30:           q.enqueue((θ, e.s, n_r))
31:       end if
32:   end while
33:   for all n ∈ N do
34:       n.r^{planned} ← n.r^{processing}
35:   end for
36:
37: **Output:**
38:   return p

There can be cases where a motion service cannot be deployed on the node connected to the client. This could be because the node’s resource capacity is too small, or higher-priority services have already preempted the node’s resources. In such cases, the next node for service deployment must be found. The nodes that have already been considered are excluded, and the next-closest node to the client is searched for producing the least traffic, referred to as the review node. If no review node exists, it means the service cannot be deployed in the edge network, and the deployment plan fails. If a review node exists, lines 28–30 verify that the resource balance with that node is considered by calculating the vector angle in the same way as before, and the tuple is added back to the queue. This process is repeated until the queue is empty, establishing a deployment plan for all services on the nodes.

As a final step, line 34 verifies that the currently processing resources for all nodes are applied to the resources reflected in the motion services deployment plan. This is used
for the next task, deploying render services. When the algorithm ends, the plan map is returned, which is reflected in the final plan map.

Once the deployment plan for all motion services of every cloud VR application has been established, the next step is to establish a plan for the render services. This can be represented by the CreateRenderDeploymentPlan algorithm. However, only changes compared to the previously discussed CreateMotionDeploymentPlan will be mentioned.

There are two main differences for render services compared to motion services deployment planning. One is that optimal deployment may not be as straightforward. Clearly, as it is performed after all motion services in the network have been deployed, considering the reduced node resources while having higher requirements, there are challenges. Additionally, the computing resources of network devices in the access network where \( n_c \) is located are generally at a lower level. The other difference is that the resulting traffic from these services to the lower services is not negligible.

While it is crucial to meet the MTP latency requirements for all users, it is equally important to efficiently manage network resources within the network to prevent issues related to latency, such as bottlenecks and network jitter. Therefore, the deployment of render services must consider both the balance of computing resources with the currently reviewed nodes and the expected traffic as metrics. Additionally, even if the user first receives a temporarily computed screen from the motion service for playback, if the render service does not update the accurately processed images a few frames later, the QoS may be compromised. When a user generates a motion input by turning their head, they first receive a temporarily computed screen from the motion service for playback, followed by the accurately processed image from the render service a few frames later. Therefore, deploying render services must consider both the computing resource balance with the current review node and the resulting traffic at the same time.

To consider two different measures simultaneously, normalization is naturally required. For minmax scaling, the minimum and maximum values of the vector angle and anticipated traffic for the service and review node are needed. This should be performed for each individual service during the first stage by calculating metrics for all nodes in the network. However, considering algorithm complexity, it can be relaxed by using appropriate constant values. For example, one could use figures like 10 for \( \min(\theta) \) and 60 for \( \max(\theta) \), where values smaller or larger than these are used as is.

3.2.2. Render Deployment Plan Algorithm

In the first stage, since the metric for each service is computed for \( n_c \), there is no need to consider the resulting traffic. Thus, the change occurs only in part of the second stage, as shown in Algorithm 3. The weight \( w \) is a constant value between 0 and 1, determining the emphasis between two planes. As lower values are ideal for both traffic and resource balance, a node with a lower-weighted sum after applying the weight is still given higher priority.

```
Algorithm 3 CreateRenderDeploymentPlan(Partial)
1: ...
2: while !q.empty() do
3: ... 
4: \( \theta \leftarrow \arccos((r_r \cdot r_s)/(|r_r||r_s|)) \)
5: \( t \leftarrow \) the expected traffic from \( n_r \) to \( n_c \) by \( \lambda, e, s \)
6: \( \lambda \leftarrow (1 - w) \ast \frac{\max(\theta) - \min(\theta)}{\max(\theta) - \min(\theta)} + w \ast \frac{\max(t) - \min(t)}{\max(t) - \min(t)} \)
7: q.enqueue((\( \lambda, e, s, n_r \))
8: ...
9: end while
10: ...
```
Once the deployment plan for render services is completed, the plan for engine services is generated. A peculiar feature of engine services is that only one exists per application, and since there are no corresponding clients, $n_c$ does not exist. Moreover, they have a moderate computing resource load, and the resulting traffic is not high, as video streaming is not involved. Therefore, the resulting traffic is not considered at all in the deployment plan for engine services. Even if deployed in the core network as a response to resource depletion or federated edge networks, the increase in synchronization latency is only slight, not significantly affecting the quality of cloud VR services.

3.2.3. Engine Deployment Plan Algorithm

The algorithm for planning the service deployment for engine services is procedurally similar to that of motion services. In the first stage of Algorithm 2, the inner angle as a resource balance metric is computed for the single engine service of each application across all nodes and is added to the priority queue.

Once the metric calculation for all engine services is completed, the nodes for deployment are decided in order of the lowest metric. Naturally, already decided services are ignored. Algorithm 4 summarizes these changes.

Algorithm 4 CreateEngineDeploymentPlan(Partial)

1: ...
2: for all $a \in A$ do
3:     for all $n \in N$ do
4:         $r_n \leftarrow [n.cpu, n.gpu, n.mem, ...]^{planned}$
5:         $r_s \leftarrow [s_a.cpu, s_a.gpu, s_a.mem, ...]^{required}$
6:         $\theta \leftarrow \arccos((r_n \cdot r_s) / (|r_n||r_s|))$
7:         $q.enqueue((\theta, s_a, n))$
8:     end for
9: end for
10: while !q.empty() do
11:     $e \leftarrow q.dequeue()$
12:     if $e.s$ is already planned to deploy then continue
13:     $r_n \leftarrow [e.n.cpu, e.n.gpu, e.n.mem, ...]^{processing}$
14:     $r_s \leftarrow [e.s.cpu, e.s.gpu, e.s.mem, ...]^{required}$
15:     if $x_i < y_i, \forall i, x_i \in r_n, y_i \in r_s$ then
16:         $p.get(e.n).add(e.s)$
17:         $e.n.r^{processing} -= e.s.r^{required}$
18:     end if
19: end while
20: if $\exists s_e, s_e$ has no plan to deploy then return -1
21: ...

4. Simulation Results

4.1. Environmental Setup

We evaluate the performance of the proposed algorithm against various metrics using simulations. To reflect as many different situations as possible, the simulation environment is set up. We implemented the simulation tool ourselves using C#, and simulations were performed using a machine with six-core 2.7 GHz Intel Core i5 processors and 32 GB of RAM.

The simulation considers a network topology consisting of a single core network with 6 nodes and a single edge network with 24 nodes. All the nodes within the network are assumed to have enough computing power to operate services, with the nodes in the core network having a higher resource capacity than those in the edge network. Lightweight HMDs are assumed to be users of one of several cloud VR applications and can be connected to any node in the edge network.
The simulation results, including the resulting traffic, are compared using figures. The computing resource balance is compared using the average standard deviation of CPU, GPU, and memory usage rates as a percentage for each node. This is determined by calculating the standard deviation of the respective resource usage rates for each node and then finding their average across all nodes.

The key simulation parameters are as follows. Each microservice’s computing resources and traffic requirements for cloud VR applications vary within the ranges specified in Table 2 for each simulation. The computing resources of network nodes vary within the ranges in Table 3 for each simulation. There are four cloud VR applications in the network, with four users per application. These figures are applied as the default parameters unless otherwise stated. Here, computing resource loads such as CPU cycle time, memory usage bytes, etc., are scaled within a certain range for ease of comparison.

### Table 2. Simulation parameters for each microservice of the cloud VR application.

<table>
<thead>
<tr>
<th>Type</th>
<th>CPU</th>
<th>Memory</th>
<th>GPU</th>
<th>Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engine</td>
<td>[10, 20]</td>
<td>[10, 40]</td>
<td>[10, 20]</td>
<td>[0.1, 1]</td>
</tr>
<tr>
<td>Render</td>
<td>[50, 80]</td>
<td>[40, 80]</td>
<td>[80, 160]</td>
<td>[20, 30]</td>
</tr>
<tr>
<td>Motion</td>
<td>[10, 20]</td>
<td>[10, 40]</td>
<td>[10, 20]</td>
<td>[7, 10]</td>
</tr>
</tbody>
</table>

### Table 3. Simulation parameters for computing resource capacity of edge nodes.

<table>
<thead>
<tr>
<th>Type</th>
<th>CPU</th>
<th>Memory</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common</td>
<td>[200, 400]</td>
<td>[250, 500]</td>
<td>[400, 800]</td>
</tr>
<tr>
<td>High-end</td>
<td>[800, 1600]</td>
<td>[1000, 2000]</td>
<td>[1600, 3200]</td>
</tr>
</tbody>
</table>

The simulation parameters are set to incorporate randomness to consider various environments, except for topology and simulation variables. To minimize the influence of these random values, each simulation result is the arithmetic mean of at least 100,000 trials.

For effective validation of the simulation, the results of using four different values of the render services deployment weight \( w \) are compared. These are represented as P0, P0.5, P0.7, and P1 in the graph, where the numeric values are \( w \). Additionally, the scheme is compared with one that applies a constraint. This scheme, indicated as ER0.7, represents the case where the proposed MSA configuration is implemented only for engine and render services with a weight \( w \) of 0.7. In this case, each render service directly processes motion input and streams the processed video, which is immediately playable by the end-user.

The proposed scheme is also analyzed and compared with our previous proposal as well as other research findings [31,32]. Our previous proposal, indicated as PVBP, provides cloud VR content to all users from a single edge server, not MSA. A single edge server per application is deployed, considering both traffic and computing resource balance simultaneously. Ref. [31] denoted the hop count (HC) as deploying a single server at the node with the least hops from the client. Ref. [32] defined the worst fit (WF) as deploying servers with lower computing resource demands on nodes with more residual resources.

### 4.2. Evaluation and Discussion

#### 4.2.1. Effect of Changes in the Number of Users per Application

Figure 4 illustrates the simulation results for the average traffic load per user, the average standard deviation of the computing resource usage per node, and the average network distance per user, varying according to the number of users per application. The network distance represents the number of hops between the node where the motion processing service is deployed and the user utilizing that service. Thus, the network distance metric can determine how close to the user the deployment should be to ensure user-specific MTP latency.
Figure 4. Changes in key network metrics with the increase in the number of users per application. (a) Traffic load per user; (b) Stdev of computing resource usage per node; (c) network distance per user.

Figure 4a shows the changes in network traffic with an increase in the number of users. The proposed scheme shows a more gradual increase at a lower level as the traffic consideration weight increases. This implies that as the number of users increases, the computing resource demands rise, increasing the likelihood of render and motion services being deployed at less-optimal locations. The PVBP, HC, and WF schemes provide content for a single application; therefore, an increase in the number of users results in a higher chance of services being deployed in the core network due to increased resource demands. In particular, the HC scheme performs well in a single-user environment in terms of traffic but struggles in a multi-user environment.
Figure 4b shows that HC and WF schemes, which do not consider computing resource balance, exhibit higher variance overall. This reflects the difficulty of matching the ratio of service resource requirements to the node’s resource capacity as the number of users in a single application service increases. P1, which does not consider resource balance when deploying render services but inherently considers it when deploying engine and motion services, shows a gradual rise. The proposed scheme, by splitting cloud VR into three services and distributing them according to their operational characteristics, alleviates the load on nodes and uses resources more evenly. This implies an increased probability of more services being deployed in better locations.

Figure 4c indirectly represents the average MTP latency per user. The single-service, multi-user accommodating schemes of PVBP, HC, and WF exhibit high network distances. MTP latency, especially from a network perspective, can be a very tight constraint, meaning that these traditional schemes struggle to effectively provide multi-user cloud VR content. ER0.7, while generating lower traffic as the render service transmits the final processed image, may have a relatively higher network distance.

4.2.2. Effects of Resource Usage Types of Applications

Figure 5 illustrates the simulation results for four types of applications: standard content, content requiring high-resolution video streaming with high traffic requirements, content demanding high-level rendering with high computing resource requirements, and content high in both aspects. The traffic requirements for the high-traffic types are set to twice that of the standard types, and the computing resource demands for high-resource types are set to 1.5 times that of standard types. This is used to evaluate the adaptability of the proposed scheme to various application types.

Figure 5a shows that PVBP, HC, and WF have similar estimated traffic for different types. For the proposed scheme with a traffic consideration weight above 0.7, the expected traffic is relatively lower than the comparison schemes. This indicates that the proposed scheme is more efficient in terms of traffic generated compared to the other schemes when providing multi-user cloud VR in the edge network. Figure 5b shows the same results as the previous simulation. The proposed scheme maintains a relatively good resource balance for all types when the traffic consideration weight is 0.7 or less. PVBP considers a traffic and resource balance of 0.5, but due to the limitations of operating a single application server, it uses computing resources in a more unbalanced way than the proposed scheme that follows MSA.

Comparing P0.7 and ER0.7, P0.7 generates about 40% more traffic than ER0.7, which is related to setting the parameter that renders the service-generated traffic to about three times higher than the motion service-generated traffic. In actual implementation, this traffic can be reduced by increasing the implementation efficiency. However, as shown in Figure 5c, from the MTP latency perspective, the proposed scheme can keep the network distance close to 0 in various situations by prioritizing the deployment of motion services with relatively low computing resource requirements near users. P0.7 shows more than 50% better performance than ER0.7, which means that the proposed scheme’s MSA configuration can more easily satisfy the MTP latency for multi-user applications.
Figure 5. Changes in key network metrics based on resource usage types of applications. (a) Traffic load per user; (b) Stdev of computing resource usage per node; (c) network distance per user.

4.2.3. Effects of Changes in Computing Resource Capacity

Figure 6 represents the results of verifying the operation of the proposed scheme according to the computing resource capacity within the network. The resource factor is used to vary the node computing resources, multiplying it by the max boundaries of each resource in Table 3. Figure 6a shows the traffic load when the resource factor increases from 1 to 4. Even with increased computing resource capacity, schemes performing single-server operations still show high average traffic per user due to the inevitable increase in the average network distance per user.
The average resource usage balance per node according to the edge network computing resources is shown in Figure 6b. Schemes that do not consider resource balance or use a single server show increased imbalance as the node’s resource capacity increases. This is due to the tendency to concentrate servers with high resource demands on one node. Furthermore, Figure 6c shows that the proposed scheme can ensure MTP latency well, regardless of the network computing resource capacity. In the case of ER0.7, the render services still failed to be deployed for \( n_c \) due to the high computing resource requirements.

In summary, even with a significant increase in the computing resource capacity of the edge network, it is difficult to substantially improve the user experience and re-
source metrics for cloud VR applications without effective MSA configuration and deployment schemes.

4.2.4. Effects of Changes in Client Locality

Figure 7 represents the simulation results for a scenario where user locality increases, forming hotspots within the network. Users are connected to specific edge nodes following a Zipf distribution [40,41]. The Zipf distribution is defined as 

\[ P(n) = \left( \frac{1}{n^s} \right) / \left( \sum_{i=1}^{N} 1/i^s \right), \]

where \( P(n) \) represents the probability of a client being placed at the nth node. Here, \( s = 0 \) corresponds to a uniform distribution, and \( s = 5 \) depicts a situation where nearly all users are connected to a single node.

Figure 7a shows the changes in network traffic per user with varying user locality. With the proposed scheme, as locality increases, so does the average network traffic per user. The proposed scheme tries to deploy motion and render services close to users, but if the hotspot node’s resources cannot accommodate all the resource demands of these services, they are deployed to adjacent nodes instead. In particular, the high traffic requirements between the render and motion services, combined with the elongation of this segment, significantly increase traffic. This can be mitigated by efficient implementation of the render-motion services. Conversely, PVBP and HC show a decrease in traffic as locality increases, suggesting they are less affected by the multi-user impact in such scenarios.

![Graph showing changes in network traffic per user](image)

(a) Traffic load per user (Mbps)

(b) Slower of resource usage per node (%)
Figure 7. Changes in key network metrics based on client locality. (a) Traffic load per user; (b) Stdev of computing resource usage per node; (c) network distance per user.

Figure 7b shows the resource balance, where extreme situations also disrupt the balance. Figure 7c represents the average MTP latency per user, and it can be seen that as the user locality becomes more extreme, the proposed scheme’s motion services are also unable to be deployed at $n_c$. In summary, as user locality becomes more extreme, the overall efficiency of the proposed scheme may decrease, which can be overcome with efficient implementation. Alternatively, if such situations persist, vertically scaling the computing resources of the affected node could be a solution.

4.2.5. Effects of Changes in Computing Resource Locality

Figure 8 represents the simulation results when assuming computing resource locality for the nodes. It demonstrates the network performance indicators when certain nodes have asymmetrically vast resources, regardless of user location. The resource locality is also based on the Zipf distribution.

As shown in Figure 8a, there are no significant changes observed in terms of traffic metrics. Even with the proposed scheme, even if the vector angle or traffic metrics are poor, specific nodes that operate all microservices may cause a slight increase in the metrics. Similarly, as shown in Figure 8b, to maintain a low MTP latency, deployment occurs at nodes with unfavorable vector angles, leading to decreased balance due to the use of all nodes. However, as shown in Figure 8c, from the perspective of MTP latency, the proposed scheme maintains a good performance without significant issues.

Figure 8. Cont.
Figure 8. Changes in key network metrics based on computing resource locality. (a) Traffic load per user; (b) Stdev of computing resource usage per node; (c) network distance per user.

5. Conclusions and Future Work

Cloud VR faces challenges such as MTP latency issues due to physical distance from server and network bandwidth issues associated with the transmission of large-volume VR content. These can be addressed by providing content in an edge network with sufficient computing and networking resources. However, in a multi-user environment, it remains a challenge to satisfy the MTP latency requirements for all users with a single server. To address this, our study proposes a provisioning scheme that guarantees MTP latency below a certain level for multi-user cloud VR while ensuring balanced use of edge network computing resources. The proposed scheme structures cloud VR applications into engine, render, and motion services using the MSA concept and establishes service deployment plans considering the characteristics of each service. The plans not only ensure MTP latency but also aim to reduce the waste of networking and computing resources.

The proposed scheme has been evaluated from various perspectives compared to other research through simulations. Various scenarios were considered in the simulations, and it was observed that the proposed scheme effectively responds to the multi-user environment in terms of network congestion, balanced use of computing resources, and MTP latency. Implementing the proposed scheme would allow the users of multi-user cloud VR to seamlessly access content using their lightweight HMDs while reducing the impact on the edge network.

Future research aims to study deployment algorithms for operating cloud VR and gaming in federated edge resource environments. This means considering situations that extend beyond environments like company campuses or local networks, such as those spanning across wider areas like Tier-3 ISP networks. Finally, to address future extended
realistic support issues, we plan to research the migration of cloud VR microservices for users wearing HMDs with passsthrough views and having mobility in outdoor environments.


**Funding:** This work was supported by the Korean Planning & Evaluation Institute of Industrial Technology and funded by the Korean government (Ministry of Trade, Industry and Energy, Ministry of Science and ICT, Ministry of the Interior and Safety, National Fire Agency, Project Number: 1761028602). Also, this work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) under the Artificial Intelligence Convergence Innovation Human Resources Development (IITP-2024-RS-2023-00254177) grant funded by the Korea government (MSIT).

**Data Availability Statement:** Data is contained within the article.

**Conflicts of Interest:** The authors declare no conflicts of interest.

**References**


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.