This article has been accepted for publication in IEEE Network. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MNET.2023.3320440

SCAXR: Empowering Scalable Multi-User Interaction for Heterogenous XR Devices

Yakun Huang, Haowen Wang, Xiuquan Qiao, Xiang Su, *Member, IEEE,* Yang Li, Schahram Dustdar, *Fellow, IEEE,* Ping Zhang, *Fellow, IEEE*

***Abstract*—Scalable multi-user interactions on diverse extended reality (XR) devices are vital for the metaverse's fruition. However, issues like broad user access, intensive interaction rendering, and limited device resources complicate existing interactions based on client-server and peer-to-peer structures. The metaverse's demands for scalable access and detailed scene rendering intensify these problems. In response, we present SCAXR, a collaborative architecture enhancing multi-user interaction. SCAXR leverages three key components: an on-demand rendering module, a distributed rendering process, and edge-cloud synchronization. This module ensures timely communication between XR devices and edge servers. We tested SCAXR's efficacy with a Unity Render Streaming-based XR meeting prototype. Results show SCAXR boosts access capacity by 50% over traditional methods and enhances complex scene interaction performance by up to 7.8 times in rendering frequency.**

I. Introduction

REAL-TIME communications and collaborations are fundamental for the implementation of multi-user extended reality (XR) within the metaverse era, allowing XR users to interact and exchange their states and operations within a unified virtual environment [1], [2]. The metaverse necessitates fine-grained and large-scale interaction scene rendering computation coupled with intensive interaction data communications among users (e.g., displacement of coordinates of virtual objects, users' viewport changes, and controller clicks). As the user base for metaverse interactions grows, numbers could soar to hundreds or even thousands. Yet, current research [3], [4] mainly tests with three to five devices. As access scales, intricate scenes, and virtual objects multiply, XR devices with limited resources will face tougher rendering tasks like vision segmentation and recognition. This presents two primary challenges: dynamically scaling user access and ensuring intense multi-user rendering across varied XR devices with resource constraints.

The multi-user interaction architectures are detailed in Fig. 1, encompassing client-server, peer-to-peer, and distributed server approaches. Within the client-server architecture, servers cater to XR device requests by hosting, delivering, and managing resources. Two interaction styles, depicted in Fig. 1(a), are client-server-C and client-server-L. The former is server-centric for synchronization and dense rendering, while the latter leans on devices for intensive rendering, using servers only for synchronization. Weber *et al.* [3] introduced a modular multi-user XR framework adaptable to diverse applications. Yet, their testing was limited to three devices and a basic AR interaction model. ARENA [4] presents an edge architecture for easily creating collaborative XR applications on WebXR browsers. Its real-world applications, though, are centered on simple VR chats and AR models with only three devices. Additionally, ILLIXR [5] pinpoints key factors influencing XR systems, such as latency, rendering frequency, and resource handling.

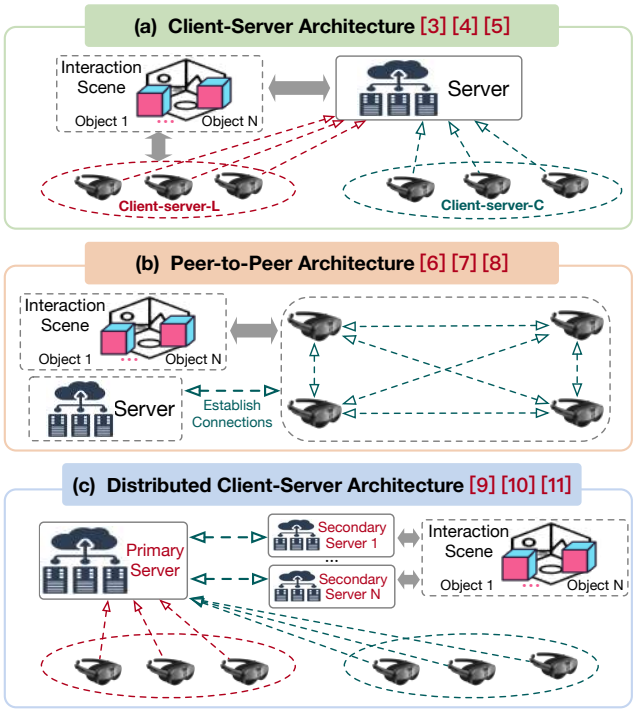

Fig. 1. Scheme of multi-user interaction architectures. (a) In the client-server-C, the server fully handles synchronization and rendering, while in the client-server-L, the server is used solely to synchronize operations and static scenes and objects. (b) This method illustrates each device bearing complete responsibility for communication and rendering. (c) The primary server oversees interactions and delegates rendering tasks to secondary servers.

In the peer-to-peer architecture, devices directly exchange and synchronize interactions, such as location and dynamic virtual objects [6]. While advantageous in confined interaction ranges, as shown in Fig. 1(b), each of the four devices first synchronizes interactions from peers and then conducts local

Y. Huang, H. Wang, X. Qiao and P. Zhang are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China. E-mail:{ykhuang, hw.wang, qiaoxq, pzhang}@bupt.edu.cn.

X. Su is with the Department of Computer Science, Norwegian University of Science and Technology, 2815 Gjøvik, Norway. Email:xiang.su@ntnu.no.

Y. Li is with the Department of Service Research, China Mobile Communications Research Institute Bejing 100053, China. E-mail: liyangyw@chinamobie.com.

S. Dustdar is with the Distributed Systems Group, Technische Universität Wien, 1040 Vienna, Austria. E-mail:dustdar@dsg.tuwien.ac.at.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on November 15,2023 at 07:16:35 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Network. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/MNET.2023.3320440

rendering. However, this architecture demands devices with robust computing and a reliable network to ensure consistent inter-device experiences [7]. Any unstable connection or underperforming device can disrupt synchronization, jeopardizing interaction consistency. Such disruptions can cause delays, notably in video games or intensive AR applications [8]. Hence, it's best suited for a limited user group in a localized setting.

The distributed architecture, an evolution of the client-server model, uses multiple servers to support rendering complex scenes and virtual objects for a vast array of devices [9]. A primary server directs intensive interaction and rendering tasks to secondary servers [10]. While inheriting data synchronization from its predecessor, it still grapples with network communication latency issues [11]. SEAR [12] offers a cache strategy for multi-user AR's computer vision tasks. Yet, this architecture doesn't fully harness resources from devices with diverse computational capabilities. The model mitigates computational strain by partitioning the interaction computation (i.e., the algorithms and computational methods we employ to process and understand this data. e.g., pose estimation, coordinate transformation) and allocating it across servers. However, existing multi-user XR interaction architectures still struggle to balance rendering loads dynamically between servers and resource-limited devices.

In this work, we propose SCAXR, a scalable multi-user interaction architecture for resource-constrained XR devices. Our approach develops an on-demand and distributed rendering mechanism to improve the scalability of the existing cloud-edge-device architecture. Specifically, the on-demand rendering module can dynamically distribute intensive rendering tasks to each device based on their computing resources and server conditions. Subsequently, we propose distributed rendering and edge-cloud collaboration mechanisms to ensure synchronized access and data consistency for multiple users. We have implemented a prototype of a multi-user XR meeting using the Unity Render Streaming framework[1] and evaluated its performance based on the motion-to-picture latency, synchronization latency, GPU load, and in-depth analysis with different XR devices. Our results demonstrate an improvement in scalability when accessing devices of different capabilities and expand the maximum access capability of interactions on the server. Moreover, SCAXR can support interaction devices with different computing capabilities, provide lower motion-to-picture latency, and improve the rendering frequency compared to baseline methods.

The paper is structured as follows: Section II discusses the motivation and challenges of multi-user XR interaction; Section III introduces the proposed SCAXR framework and its components; Section IV presents the prototype system and evaluates the performance of SCAXR; and Section V concludes the paper and delineates future directions.

II. Motivation and Challenges

To support varying user access scales, we use Selenium[2], a tool that autonomously manages user interactions, capable of handling dozens of users on standard computing devices. Our multi-user interaction application is defined by a shared scene (78MB) and three dynamic objects, each around 14MB. Evaluating scalability (see Fig. 2), we adopt typical client-server and peer-to-peer architectures on an AliCloud server with a T4 card GN6i GPU. Specifically, the client-server architecture has (1) Client-server-C: The server handles synchronization and rendering for each device, sending rendering results as video streams; (2) Client-server-L: The server synchronizes the scene, objects, and device states, but each device locally renders results. Meanwhile, the peer-to-peer method first connects devices, then shares the scene and objects among them. All synchronization and rendering processes are entirely device-centric.

Fig. 2(a) and Fig. 2(b) illustrate the effects of rising numbers of access devices on the edge server's synchronization latency and GPU load, respectively. Synchronization latency here refers to the alignment of user operations, states, and rendered outcomes, excluding rendering and computational latencies. Using Selenium to simulate multi-user access, a laptop's capacity to support users is finite, maxing out at 20 users in our tests. Results highlight an increase in both synchronization latency and GPU load proportional to user count. Notably, with five access users, client-server-L and peer-to-peer architectures nearly deplete all GPU resources for local rendering. This suggests that the scalability of these methods is restricted by the device's computational prowess. In our tests, the laptop could accommodate five simulated access users. Further, while raising the number of users didn't markedly raise synchronization latency for the client-server-L and peer-to-peer methods, it did max out the GPU, leading to an average rendering frequency below 1Hz (as per the Unity engine). The rendering frequency refers to the number of frames rendered per second after the device has obtained synchronized data.

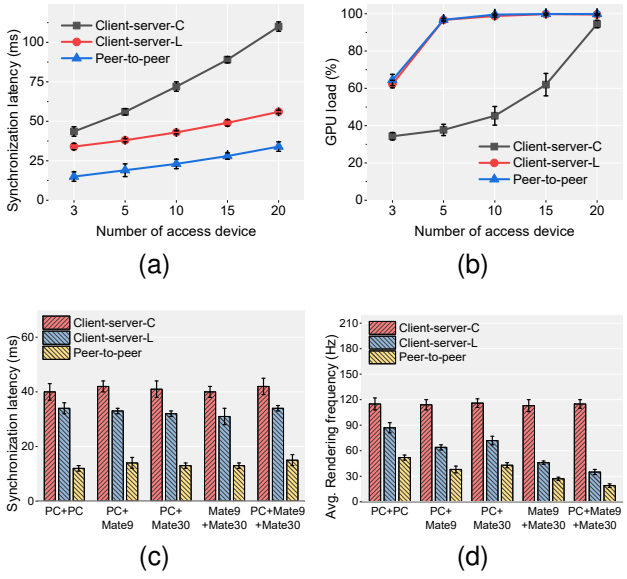

Fig. 2. Interaction using access devices of varying scales and computing capabilities: (a) synchronization latency across varying scales. (b) GPU load in correlation to different scales. (c) synchronization latency relative to devices with distinct computing capabilities. (d) comparing average rendering frequency among devices with differing computing capabilities.

[1]https://github.com/Unity-Technologies/UnityRenderStreaming
[2]https://www.selenium.dev

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on November 15,2023 at 07:16:35 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Network. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/MNET.2023.3320440

Our study utilized four varied devices: two laptops and two smartphones - HUAWEI Mate30 (with Mali-G76 GPU) and Mate9 (with Mali-G71MP8 GPU) - aiming to appraise the interaction performance across varying computational abilities. As depicted in Fig. 2(c) and Fig. 2(d), synchronization latency and the average interaction rendering frequency are illustrated, respectively. Interestingly, variations in device computational capacities barely influenced data synchronization, such as scenes, objects, and states. These aspects are mainly steered by network performance. Devices with distinct computational capabilities demonstrated lower rendering frequencies under client-server-L and peer-to-peer models. This is governed by the rendering capabilities of the device and the server. Conversely, the client-server-C model exhibited substantially higher rendering frequencies as all rendering activities transpire on the cloud server, outstripping the local device's capabilities. Notably, the peer-to-peer model achieved the smallest synchronization latency, given that most device synchronization is contingent upon the WiFi network. Our primary pursuit is to bolster rendering efficacy while curbing the impact of a range of devices with diverse computational capabilities.

Our preliminary analysis highlights the limitations of the existing client-server and peer-to-peer architectures in terms of scalability between heterogeneous devices. To actualize optimal interactions, two paramount challenges arise:

(a) Large-Scale XR User Access: Dynamics and Scalability. Conventional multi-user interaction frameworks are tailored for smaller, localized interaction domains, consequently limiting scalability. Contrastingly, emerging metaverse platforms could necessitate expansive XR interactions that traverse varied geographical locations, exacerbating interaction latency and intensifying rendering computations among devices. The concurrent access by a plethora of users places significant demands on both networking and computational capacities. Key hurdles include optimizing cloud resources and mobile energy consumption, all while upholding stringent standards for interaction latency and data fidelity. The dynamic engagement of mobile XR users also necessitates adept resource distribution and task scheduling.

(b) Heterogeneous and Resource-Constrained XR Devices: Facing Intensive Rendering and AI Computations. XR interaction services heavily depend on 3D visual computing and immersive rendering computations, posing significant challenges for both cloud centers and XR devices [13]. Achieving a full XR interaction implementation is arduous due to the massive computational requirements either in the cloud or on devices. For instance, 3D scene models within the Unity rendering engine can demand hundreds of megabytes or even gigabytes. AI computations, such as those computer vision tasks based on DNN and CNN structures (e.g., object recognition, tracking [12] and 3D generation computations [14]), require GPU support for real-time inference. The extensive computation required by numerous users limits the accessibility of XR devices.

III. SCAXR Architecture

We introduce the SCAXR architecture and its components, illustrated in Fig. 3. Rooted in the existing cloud-edge-device framework [2], SCAXR offers a scalable solution. The on-demand rendering module in SCAXR ensures efficient computation by dynamically distributing rendering tasks between XR devices and the edge server, maximizing device computational resources. Meanwhile, our distributed rendering mechanism caters to a vast number of XR users. Although the on-demand rendering offloads tasks from the edge server to XR devices to minimize the server's computational load, the addition of multiple devices necessitates multiple edge servers for synchronized interaction. The cloud center becomes pivotal for synchronizing user actions and scene states across vast distances and different edge servers. During the start, scenes and static objects are duplicated over edge servers, removing the need for inter-edge result transfers. Consequently, edge-cloud synchronization maintains long-distance interaction data consistency.

On-demand Rendering Module. As illustrated in the left of Fig. 3, this module dynamically delegates rendering tasks, considering the computing capacity of XR devices and edge servers. The key steps involved include state verification and synchronization, scheduling, interaction updates, and the transfer of rendering results. Initially, we represent the rendered scene within a three-dimensional vector space, where each object is symbolized by a 3D coordinate. Assume that there are N devices, represented as $C_1, C_2, ..., C_N$, and M rendering objects within the scene, symbolized as $O_1, O_2, ..., O_M$. Each object, O_i, has its position represented by the 3D vector (x_i, y_i, z_i). Subsequently, the on-demand rendering process unfolds as follows:

STEP. 1. State Verification and Synchronization. Initially, the edge server E verifies the access state of all devices and obtains their performance weights, denoted by $W = \{w_1, w_2, ..., w_N\}$. Each w_i indicates the computing capability and network condition of a device. Specifically, w_i is determined as $w_i = (1-\delta) + B_c/B_o$, where δ represents the GPU usage, and B_c and B_o stand for the current and optimal network bandwidths, respectively. This formulation can be adjusted or substituted with other parameters. Subsequently, the edge server broadcasts the scene and objects to all devices, achieving primary synchronization.

STEP. 2. Rendering Scheduling. The edge server E collaboratively determines rendering task distribution with devices, considering their performance and network conditions. For less capable devices, E might handle the rendering of complex scene objects, leaving simpler ones to the device. In contrast, high-performance devices might receive more rendering tasks, easing the server's burden. Task assignments are based on a device's performance weight W_i, expressed as $R(E, C_i, W_i)$. The collection of objects rendered by E for device C_i is given by R_i.

STEP. 3. Interaction Updates. XR devices send their user interaction states, such as location and activity, to the edge server E. In response, E revises the scene and objects, and disseminates the updates to all devices. If device C_i has an interaction state I_i, it communicates this as $U(C_i, I_i)$ to E. The server then modifies the scene object using this data, represented as $U(E, \{I_1, I_2, ..., I_N\})$. Afterward, E broadcasts the refreshed scene information to all devices, denoted as

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on November 15,2023 at 07:16:35 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Network. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MNET.2023.3320440

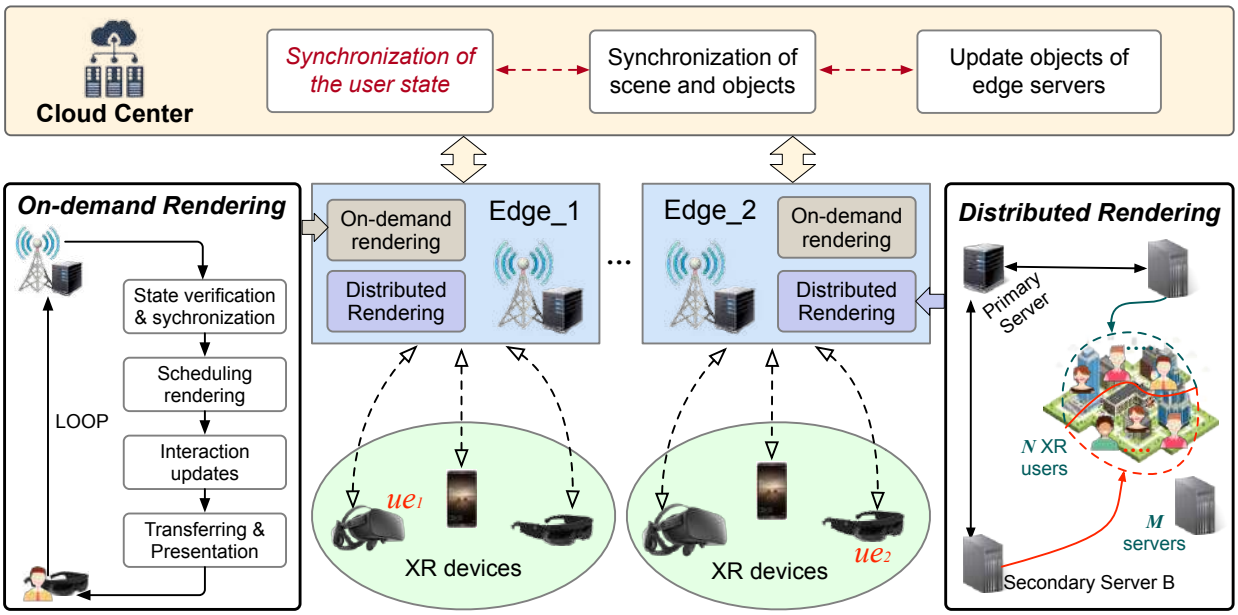

Fig. 3. The SCAXR architecture for multi-User XR interaction integrates three primary modules: On-Demand Rendering, Distributed Rendering, and Edge-Cloud Synchronization, facilitating a robust framework for complex interaction scenarios.

$B(E, \{O'_1, O'_2, ..., O'_M\})$.

STEP. 4: Transfer and Presentation of Results. The edge server transmits the rendering outputs, like images and textures, to the device. This device then assimilates results with its local renderings and showcases them to the user.

By repetitively executing the outlined steps, adaptive rendering is attained, ensuring seamless interaction between the edge server and XR devices. This process adapts to ever-changing network conditions and computing capacities.

The scheduling of on-demand rendering in STEP. 2 is informed by the present conditions of the access device, which include network latency, computing capabilities (e.g., GPU), and the current load, in addition to the complexity of the task. Task complexity is determined by factors such as scene complexity, texture size, and lighting calculations. It can also be extended to include more characteristics. A dynamic allocation method is established with the primary objective of reducing total rendering latency while ensuring balanced load distribution. The assignment of rendering tasks, including scenes and other objects, is denoted as a two-dimensional matrix $T[i][j]$, representing the weight of the i^{th} device allocated to the j^{th} rendering task. The optimization objective seeks to minimize the expression $\sum(W[i][j] * (network\_delay[i] + task\_complexity[j]))$, adhering to constraints that ensure task completion. Firstly, each rendering task is assigned exclusively to a single device or edge server, denoted as $(\sum(W[i][j]) = 1, \forall j)$. Secondly, the total load per device cannot surpass a predetermined threshold: $(\sum(W[i][j] * task\_complexity[j]) <= max\_load * available\_resources[i], \forall i)$. This optimization model can be resolved using a variety of optimization algorithms, such as linear programming.

Moreover, the rendering outputs from both the edge server (denoted as C_{edge} and D_{edge} for the color buffer and depth buffer respectively) and the XR device in STEP. 4 (denoted as C_{local} and D_{local} for the local color buffer and depth buffer respectively) are integrated using depth value comparison. A novel color buffer, C_{final}, is introduced to store the resulting merged image. Each pixel (i, j) of C_{edge} and C_{local} is then processed as follows: (a) Depth values of the edge server and local rendering results are obtained: $depth_{edge} = D_{edge}[i][j]$, $depth_{local} = D_{local}[i][j]$. (b) The two depth values are compared: if $depth_{edge} < depth_{local}$, then the edge server's color rendering is utilized: $C_{final}[i][j] = C_{edge}[i][j]$. Otherwise, the local rendering's color value is used: $C_{final}[i][j] = C_{local}[i][j]$. Upon concluding this pixel-wise operation, C_{final} emerges as the merged rendering buffer. The final image render is achieved by presenting C_{final} to the user. This depth-dependent image synthesis technique effectively merges the edge server and local device rendering outputs, furnishing a final rendering for the user.

Distributed Rendering Mechanism. The distributed rendering framework utilizes a hierarchical architecture composed of one primary server and multiple secondary servers, designed to enable real-time synchronization and efficient rendering services in high-density device environments. In Fig. 3, the primary server has three principal functions: (a) collecting status and interaction data from all devices; (b) measuring the intricacy of the current rendering tasks for XR devices and distributing them to secondary servers based on their capability and workload; (c) sustaining updated task assignments via regular communication with the secondary servers. On the other hand, the responsibilities of the secondary servers consist of (i) commencing the execution of tasks using the on-demand rendering method upon receipt of the corresponding assignments from the primary server; (ii) regularly communicating with the primary server to convey the device connection statuses and receive updates on assigned tasks.

We describe in detail how the distributed mechanism manages interaction and rendering tasks during large-scale usage. The interaction rendering pertains to the graphical display or

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on November 15,2023 at 07:16:35 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Network. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/MNET.2023.3320440

visualization of the computed interactions. In the multi-user XR interaction context shown in Fig. 3, the primary server manages states across devices and assigns tasks to secondary servers based on rendering needs. The allocation process comprises three stages: **(a) Primary Server Assignment.** For simultaneous XR user access, devices are allocated to secondary servers via a weighted Round Robin load-balancing strategy, factoring in user location, device capabilities, and rendering complexity. **(b) Rendering and Interaction.** Devices liaise with their assigned secondary servers for task rendering, utilizing on-demand techniques to optimize performance. **(c) Data Synchronization and Updates.** To maintain consistent rendering, the system synchronizes shared data like scene specifics and user operations across secondary servers. This strategy optimizes server load, enhances scalability based on user numbers and computational demands, and ensures efficient, high-quality rendering.

In the dynamic allocation of secondary servers, we employ the Weighted Round Robin algorithm. Servers are weighted based on their computing capabilities and network conditions, with a higher weight indicating a better ability to handle more requests. We initialize a variable, $current\_weight$, to zero and assign initial weights to each server. An index variable tracks the chosen server. For each access device, the process involves: (a) finding the server with the highest $current\_weight$, (b) designating this server for the task, and (c) adjusting its $current\_weight$ by subtracting the collective initial weights of all servers. Subsequently, each server's $current\_weight$ is incremented by its initial weight. If all server weights drop below zero, they revert to their starting values, restarting the allocation. This ensures optimal server assignment for rendering tasks.

In densely populated XR environments, users interact in shared virtual spaces via on-demand and distributed rendering. When users connect to distant edge servers, edge-cloud collaboration becomes vital for synchronizing data like location, actions, and object states. For example, as depicted in Fig. 3, XR users ue_1 and ue_2 link to $edge_1$ and $edge_2$, respectively. The synchronization and interaction across these users, bridging vast distances and different server domains, involves: (a) Cloud-mediated data synchronization between $edge_1$ and $edge_2$, encompassing scene components, objects, textures, etc. (b) Exchange of user states—$edge_1$ conveys ue_1's status to the cloud, which then syncs it to $edge_2$, and the reverse occurs for ue_2. Conclusively, $edge_1$ and $edge_2$ render the respective users' environments, leveraging on-demand techniques based on the synchronized user data and scene details.

IV. Prototype and Evaluation

A. Use case and Implementation

Fig. 4 depicts the SCAXR-driven multi-user XR meeting service, enabling interactions across various devices. The cloud server coordinates edge servers, ensuring consistent scene, object, and interaction states. The journey starts with the conference initiator setting up a meeting room on an edge server, allowing various XR users to join. The setup harnesses an AliCloud server as the cloud node, two high-end servers

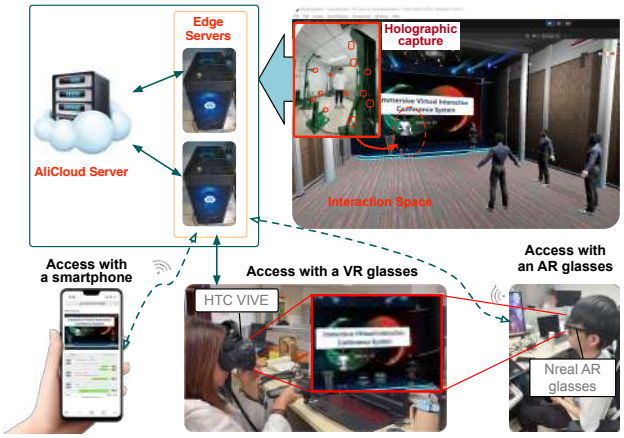

Fig. 4. Use case and the prototype system of a multi-user XR meeting.

(with NVIDIA 3060 GPU) as edge servers, and multiple XR devices. Unity Render Streaming handles rendering, and WebRTC manages XR device connections and edge server data synchronization. Cloud server synchronization interfaces across different edges are crafted via Python Flask. Our platform supports AR glasses, VR glasses, and smartphones. It also integrates real-time holographic volumetric video capture [13], [15] to amplify the immersive quality of interactions. Given the range of XR devices, the on-demand rendering module at the edge servers provides tailored rendering services. For instance, while VR and mobile devices get a full render of the virtual meeting, AR glasses just get the 3D holographic speaker, blending it into the actual environment. These merged renderings are then distributed to the respective XR devices through the Unity Render Streaming framework.

B. Experimental settings

In Fig. 4, we delineate the network environment specifications of SCAXR. Utilizing China Unicom's 5G network at BUPT, we set up two edge servers, maintaining a network latency of 5ms between them. All XR devices synchronize with these edge servers over a WiFi connection adhering to the IEEE 802.11ac standard, capable of reaching data rate speeds up to 1.3 Gbps in the 5 GHz frequency band. Nonetheless, during our tests with multiple devices under diverse conditions, the average throughput approximated 430 Mbps. Moreover, the mean end-to-end latencies for edge-cloud and device-edge interactions stand at 20 ms and 8 ms, respectively

C. Evaluation

We benchmark the scalability of SCAXR against traditional client-server and peer-to-peer methods, evaluating motion-to-picture and synchronization latency, and average GPU load. The motion-to-picture metric quantifies the latency all users encounter following an action initiated by any user. 'client-cloud' and 'client-edge' refer to server deployments at distinct layers. 'client-cloud/edge-C' and 'client-cloud/edge-L' indicate where rendering computations are carried out, either on the server or the accessing devices. For interaction scenarios, we employ a scene and objects amassing 134MB in total size.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on November 15,2023 at 07:16:35 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Network. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/MNET.2023.3320440

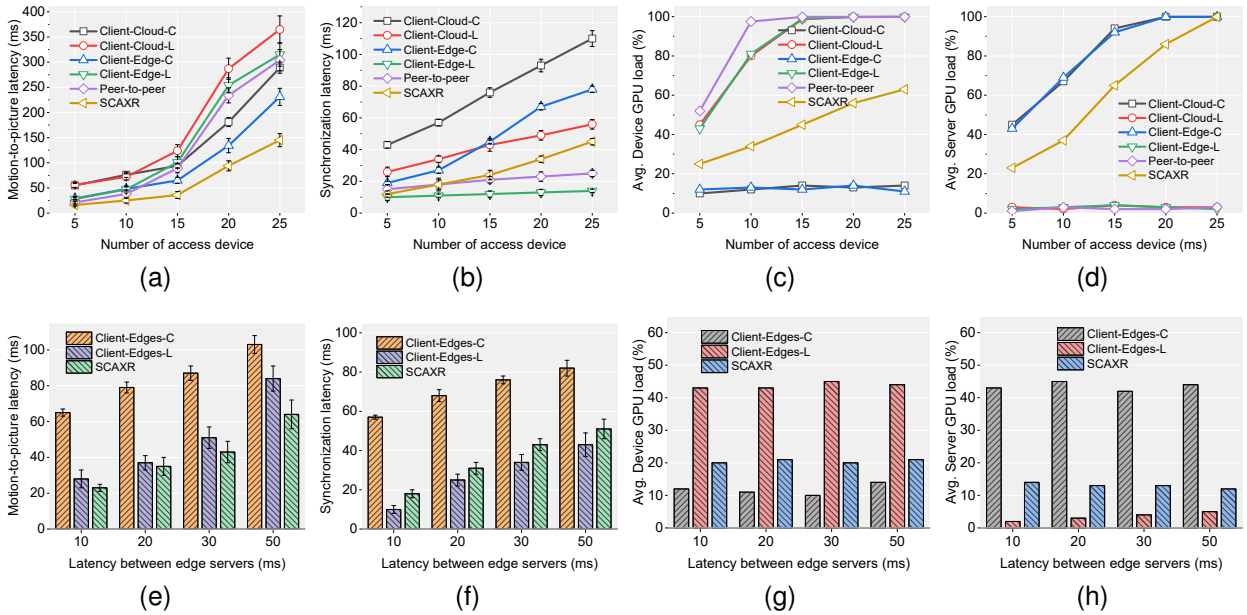

Fig. 5. Comparison of SCAXR and baseline architectures. (a) Single-server motion-to-picture latency; (b) Single-server synchronization latency; (c) Average GPU load of access devices (one server); (d) Average GPU load of server (one server); (e) Dual-Server motion-to-picture latency; (f) Dual-server synchronization latency; (g) Average GPU load of access devices (two servers); (h) Average GPU load of server (two servers).

Scene complexity is gauged by the count of its triangular faces. Using Selenium, we emulate varying user access scales, with all participants entering the same scene from random starting points to observe and interact with objects.

Results indicate the following: (1) Fig. 5(a) illustrates motion-to-picture latency, while Fig.5(b) presents synchronization latency under varied user scales on a single edge server. SCAXR notably surpasses the baseline in motion-to-picture latency, showing marked enhancement with 25 devices. At ten devices, both client-cloud-C and cloud-client-L's motion-to-picture latencies exceed 50ms, slightly impacting user experience. However, SCAXR's latency remains below 50ms even with 15 devices. As depicted in Fig. 5(b), cloud-edge-L and peer-to-peer exhibit lower synchronization latency than SCAXR. However, their motion-to-picture latency is at least 2.4 times greater than that of SCAXR when accessing 15 devices. This highlights SCAXR's efficient on-demand rendering module, balancing rendering tasks between servers and devices for optimal performance. With over 15 devices, SCAXR's latency goes beyond 50ms, reaching its limit at 20, suggesting a single server's capacity to support around 15 devices. SCAXR enhances access capacity by at least 50% over client-server methods with a latency cap of 50 ms. As interactive scene complexity grows, the supported device count might decline. Still, SCAXR proves its scalability, accommodating more devices even with increased intricacy.

(2) Fig. 5(c) and Fig. 5(d) depict the average GPU load on access devices and the edge server, respectively. Both peer-to-peer and server-only strategies encounter GPU loads exceeding 80% with just ten devices due to the entire rendering demand of the complete scene and dynamic objects by each device. Contrarily, SCAXR dynamically distributes rendering tasks, allowing the server to manage intricate scene rendering. Consequently, SCAXR's server GPU load is only 37%, while client-cloud-C and client-edge-C methods are near 65%, as shown in Fig. 5(d). With 20 devices, the average GPU load climbs to 85%, indicating the peak capacity of current resources. In summary, SCAXR consistently surpasses competing methods in motion-to-picture latency and GPU load, showcasing superior resource efficiency and scalability.

(3) In Fig. 5(e) through Fig. 5(h), we detail the performance metrics—motion-to-picture, synchronization latency, and average GPU usage—of different techniques employing two edge servers for interaction. The extended client-server architectures, denoted as 'client-edges-C' and 'client-edges-L', follow SCAXR's distributed strategy for multi-server interaction. We manipulated transmission latency between the servers to emulate varying distances. Each server was designated to either three or two of the five access devices, all with similar computational capabilities, arranged through Selenium. The interaction scene and settings remained consistent with earlier experiments. As the inter-server transmission latency rises, so do both motion-to-picture and synchronization latencies. At a transmission latency of 10ms, SCAXR and client-edges-L register a motion-to-picture latency near 20ms. The brevity in SCAXR's latency arises due to its limited reliance on server synchronization for interaction data. Conversely, client-edges-C mandates full interaction synchronization across servers prior to rendering, resulting in latencies surpassing 60ms. The chief source of synchronization latency is the interaction data from diverse server users. GPU results from Fig. 5(g) and Fig. 5(h) highlight that SCAXR's distributed rendering doesn't intensify GPU demand on devices or servers. Only interaction state data syncs on devices under the same server, while static scenes and objects sync via the cloud. Remarkably, with two servers, SCAXR averages a mere 13% GPU load (at 10ms transmission latency), a noticeable decrease compared to a single server's load for five devices shown in Fig. 5(d).

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on November 15,2023 at 07:16:35 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Network. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/MNET.2023.3320440

Consequently, the data imply that SCAXR could effectively harness additional servers to accommodate even larger user scales, possibly facilitating multi-user interaction spanning hundreds to thousands of participants.

In Fig. 6, we assess the interaction experience using access devices of varied computational strengths. The interaction content encompasses both the scene and dynamic objects, inclusive of holographic content backed by AI computation. Our experimental setup, depicted in Fig. 4, features an AliCloud server and two edge servers located across dual campuses of BUPT, separated by 20 km. Each edge server boasts 32GB RAM and an NVIDIA 3060 GPU. A chosen scene model, integrating 1.2 million triangles, was utilized. For latency and rendering frequency measurements, we employed a laptop with 16GB RAM and an NVIDIA 2080 GPU. Rendering frequency data, sourced from the Unity engine, underwent over ten measurement cycles to ensure dependable consistency.

The results show that: (1) Fig. 6(a) and Fig. 6(b) illustrate an evaluation of the rendering frequency and the motion-to-picture latency associated with interaction devices possessing varying computational capacities. In comparison to the client-cloud-L, client-edge-L, and peer-to-peer methods, the results in Fig. 6(a) indicate that SCAXR can enhance the rendering frequency of the low-performance HUAWEI Mate9 by 22%, 2.7%, and 52%, respectively. The client-cloud-C and client-edge-C methods, in contrast, manifest a higher rendering frequency than SCAXR, a fact attributable to the complete execution of rendering computations within the cloud server. However, when the computational performance of the device is substantial, as with an interactive device utilizing a Desktop PC, the rendering frequencies of both client-cloud-L, client-edge-L, and SCAXR surpass those of the client-cloud/edge-C methods. This outcome can be attributed to the considerable computational capacity of the device, coupled with the fact that the client-cloud/edge-C methods necessitated thrice rendering of the same content, thereby resulting in a diminished rendering frequency. Contrariwise, as depicted in Fig. 6(b), the motion-to-picture latencies of the client-cloud-C and client-edge-C displayed minimal fluctuations, whereas the other methods that were reliant on device rendering demonstrated performance commensurate with the average rendering frequency. Interestingly, SCAXR displayed a distinctive ability to dynamically alternate rendering tasks between the server and the device. As a result, SCAXR showcased exemplary performance even on the less powerful HUAWEI Mate9.

(2) Fig. 6(c) and Fig. 6(d) delve into the impact of scene model complexity, using three access devices with consistent computational abilities simulated through Selenium. Server-based methodologies, specifically client-cloud-C, and client-edge-C, are notably more influenced by scene intricacy than peer-to-peer, client-cloud-L, and client-edge-L techniques. The latter methods experience a more muted rendering frequency impact. This distinction is due to the server-based technique requiring every device to independently render both the scene and dynamic entities. Notably, in Fig. 6(c), SCAXR amplifies performance for scenes with intricate interactions, boasting a rendering frequency enhancement of up to 7.8 times. When scene complexity escalates, the server-based strategy demands

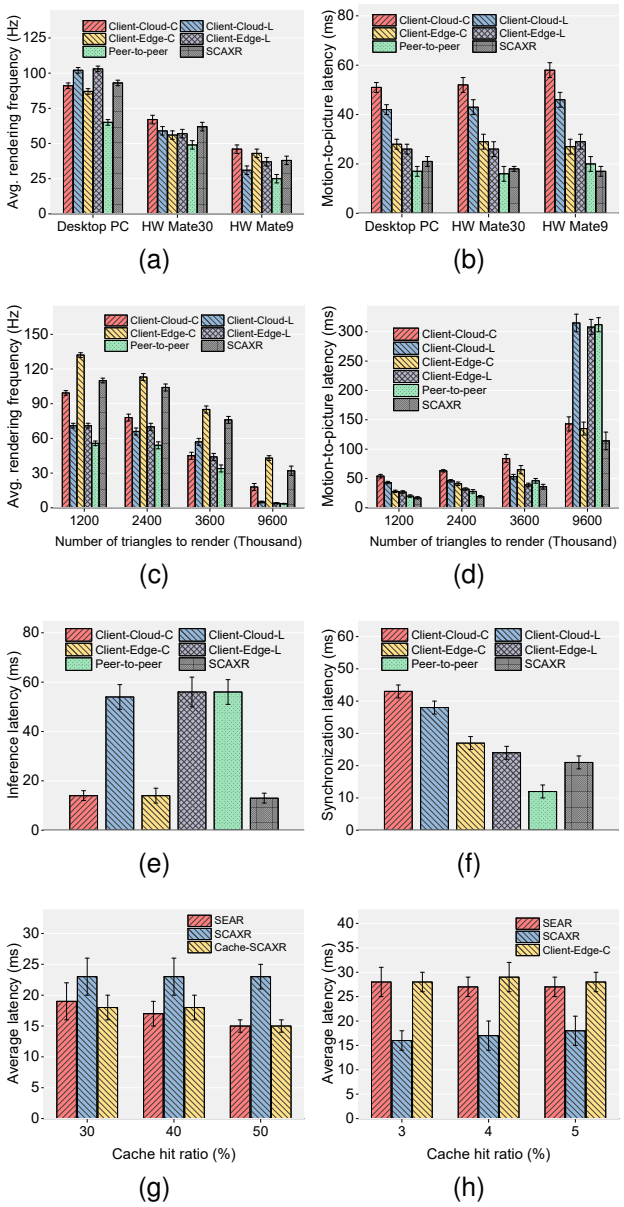

Fig. 6. In-depth analysis of SCAXR. (a) Average rendering frequency using access devices with capabilities; (b) motion-to-picture using access devices with capabilities; (c) average rendering frequency with different interaction scenes; (d) motion-to-picture with different interaction scenes; (e) inference latency of AI-based holographic computing; (f) synchronization latency of holographic contents; (g) comparing SCAXR with SEAR with on object detection; (h) comparison with cache-based SEAR on scene rendering.

rendering tasks N times more than device-exclusive rendering, where N signifies the count of interactive devices. Given SCAXR's adeptness in fluidly allocating rendering tasks between servers and devices, it consistently outperforms both approaches.

(3) Fig. 6(e) and Fig. 6(f) detail the computational expenses of rendering holographic content in SCAXR using AI methods. As illustrated in Fig. 4, real-time holographic video is first captured at the acquisition server. Here, it undergoes AI-enhanced fusion before being sent to the server for further rendering. In setups such as client-cloud-C, client-edge-C, and SCAXR, we treat the holographic content as a core part of the scene model, meaning the server directly decodes this dense

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on November 15,2023 at 07:16:35 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Network. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/MNET.2023.3320440

content. On the other hand, for configurations like client-edge-L, client-cloud-L, and peer-to-peer, we adopt the AI-based holographic video transmission method from our previous research [13]. This method transmits the video's key points and then reconstructs it based on these points at the target devices. Specifically, Fig. 6(e) illustrates the overhead associated with AI-based holographic content decoding using a Generative Adversarial Network (GAN)-based method [14].

The analysis shows that in the configurations of client-cloud-C, client-edge-C, and SCAXR, AI inference occurs server-side, resulting in a modest inference cost of around 10ms. In contrast, methods like client-cloud-L, client-edge-L, and peer-to-peer experience a much higher inference latency, surpassing 45ms. While using a compressed AI decoding model might reduce this to 15ms, it comes at the expense of decreased accuracy in the decoded holographic content. Rendering holographic content directly on access devices significantly heightens the computational load. SCAXR's on-demand rendering approach enables holographic content decoding at the edge server, which is then combined with device rendering outcomes. This highlights SCAXR's capability in handling complex and compute-intensive 3D content, including holography. Fig. 6(f) displays synchronization overhead for holographic content. Notably, there's no marked increase in synchronization overhead when transmitting key points of holographic content versus the full point cloud data. This suggests that using an AI-driven transmission method can significantly reduce data transmission volume when dealing with holographic content.

In Fig. 6(g) and Fig. 6(h), we evaluate SCAXR against SEAR [12], a caching-based multi-user AR interaction method, assessing average interaction latency for AI object detection and scene rendering tasks. Experimental conditions and device settings are consistent with prior descriptions. For Fig. 6(g), we tested using five devices on lightweight object detection tasks. Each device shares its results with others. We examined cache hit ratios of 30%, 40%, and 50%. SCAXR does not use caching, while Cache-SCAXR adopts SEAR's strategy. The metric is the average delay for devices to synchronize results. Fig. 6(h) contrasts SEAR and SCAXR on a previously mentioned scene rendering task. The findings are: (1) caching enhances AI or reusable XR computations. At a 50% cache hit ratio, SEAR and Cache-SCAXR improved by 34.78% and 30.43%, respectively. Caching reduces AI re-inferencing overhead but demands result synchronization, introducing extra latency with server-based methods. (2) SEAR's performance gain is minimal in diverse XR scene rendering. Minor device orientation or viewpoint shifts necessitate unique rendering, making caching effective only in shared or distant view scenarios. Hence, SEAR's cache hit rate is minimal, rivaling the Client-Edge-L method in such contexts. In conclusion, while caching strategies can enhance the system's performance in specific task scenarios, it's imperative to decouple and intricately design the system's computational tasks and processes to achieve tangible benefits.

V. Discussion and Conclusion

This paper presents SCAXR, a scalable multi-user XR interaction framework leveraging the synergistic benefits of cloud, edge, and device collaboration. SCAXR offers two pivotal enhancements over existing methods. First, it features a low-latency, edge-based rendering structure paired with a central cloud hub. This facilitates synchronized interaction data exchange among distant edges, thus boosting scalability. Second, SCAXR's dynamic rendering strategy adeptly mitigates the resource limitations of XR devices, offering tailored content rendering solutions that meet the diverse demands of XR users. To validate its effectiveness, we implemented a prototype of SCAXR and compared it with prevailing interaction frameworks. Employing a multi-user XR meeting as a representative example, our results underscore SCAXR's promise as a formidable framework for advanced multi-user XR interactions in intricate settings. As we gaze ahead, our ambition is to refine the user experience in SCAXR further. Proposed enhancements encompass adaptive multimodal streaming transfer scheduling, efficient large-scale XR model rendering, swift XR streaming transfers, and robust synchronization and consistency guarantees for distributed rendering.

Acknowledgment

This research was supported in part by the National Key R&D Program of China under Grant 2022YFF0904304, in part by the National Natural Science Foundation of China under Grant 62202065, in part by the Project funded by China Postdoctoral Science Foundation 2022TQ0047 and 2022M710465.

References

[1] F. Tang, X. Chen, M. Zhao, and N. Kato, "The roadmap of communication and networking in 6g for the metaverse," *IEEE Wireless Communications*, pp. 1–15, 2022, doi:10.1109/MWC.019.2100721.

[2] B. Han, P. Pathak, S. Chen, and L.-F. C. Yu, "Comic: A collaborative mobile immersive computing infrastructure for conducting multi-user xr research," *IEEE Network*, pp. 1–9, 2022, doi:10.1109/MNET.126.2200385.

[3] S. Weber, L. Rudolph, S. Liedtke, C. Eichhorn, D. Dyrda, D. A. Plecher, and G. Klinker, "Frameworks enabling ubiquitous mixed reality applications across dynamically adaptable device configurations," *Frontiers in Virtual Reality*, vol. 3, pp. 1–20, 2022.

[4] N. Pereira, A. Rowe, M. W. Farb, I. Liang, E. Lu, and E. Riebling, "Arena: The augmented reality edge networking architecture," in *2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, 2021, pp. 479–488.

[5] M. Huzaifa, R. Desai, S. Grayson, X. Jiang, Y. Jing, J. Lee, F. Lu, Y. Pang, J. Ravichandran, F. Sinclair *et al.*, "Exploring extended reality with illixr: A new playground for architecture research," *arXiv preprint arXiv:2004.04643*, 2020.

[6] N. Suslov, "Livecoding. space: Towards p2p collaborative live programming environment for webxr," in *Proceedings of the Fourth International Conference on Live Coding*, 2019, pp. 1–6.

[7] H. A. Engelbrecht and J. S. Gilmore, "Pithos: Distributed storage for massive multi-user virtual environments," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 13, no. 3, pp. 1–33, 2017.

[8] P. Ren, X. Qiao, Y. Huang, L. Liu, C. Pu, S. Dustdar, and J. Chen, "Edge ar x5: An edge-assisted multi-user collaborative framework for mobile web augmented reality in 5g and beyond," *IEEE Transactions on Cloud Computing*, vol. 10, no. 4, pp. 2521–2537, 2020.

[9] Y. Okuya, N. Ladeveze, O. Gladin, C. Fleury, and P. Bourdot, "Distributed architecture for remote collaborative modification of parametric cad data," in *2018 IEEE Fourth VR International Workshop on Collaborative Virtual Environments (3DCVE)*. IEEE, 2018, pp. 1–4.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on November 15,2023 at 07:16:35 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Network. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MNET.2023.3320440

[10] J. Donkervliet, J. Cuijpers, and A. Iosup, "Dyconits: Scaling minecraft-like services through dynamically managed inconsistency," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021, pp. 126–137.
[11] S. Shen, S.-Y. Hu, A. Iosup, and D. Epema, "Area of simulation: Mechanism and architecture for multi-avatar virtual environments," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 12, no. 1, pp. 1–24, 2015.
[12] W. Zhang, B. Han, and P. Hui, "Sear: scaling experiences in multi-user augmented reality," *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 5, pp. 1982–1992, 2022.
[13] Y. Huang, Y. Zhu, X. Qiao, X. Su, S. Dustdar, and P. Zhang, "Towards holographic video communications: A promising ai-driven solution," *IEEE Communications Magazine*, vol. 60, no. 11, pp. 82–88, 2022.
[14] Y. Huang, Y. Zhu, X. Qiao, Z. Tan, and B. Bai, "Aitransfer: Progressive ai-powered transmission for real-time point cloud video streaming," in *Proceedings of the 29th ACM International Conference on Multimedia*, 2021, pp. 3989–3997.
[15] Y. Zhu, Y. Huang, X. Qiao, Z. Tan, B. Bai, H. Ma, and S. Dustdar, "A semantic-aware transmission with adaptive control scheme for volumetric video service," *IEEE Transactions on Multimedia*, 2022, doi: 10.1109/TMM.2022.3217928.

Yakun Huang is currently a Postdoctoral Researcher at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. His current research interests include volumetric video streaming, mobile computing, and augmented reality.

Haowen Wang is currently working towards a Ph.D. degree at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. His current research interests include 3D object detection and deep learning.

Xiuquan Qiao is currently a Full Professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. His current research interests include the future Internet, services computing, computer vision, distributed deep learning, augmented reality, virtual reality, and 5G networks.

Xiang Su is currently an Associate Professor with the Department of Computer Science, Norwegian University of Science and Technology, Norway, and the University of Oulu, Finland. He has extensive expertise in the Internet of Things, edge computing, mobile augmented reality, knowledge representations, and context modeling and reasoning.

Yang Li is a project manager of the Department of Service Research, China Mobile Communications Research Institute. His current research interests include 5G, augmented reality, and video streaming.

Schahram Dustdar (Fellow, IEEE) is a Full Professor of Computer Science and is heading the Distributed Systems Research Division at the TU Wien. He is an ACM Distinguished Scientist, ACM Distinguished Speaker, IEEE Fellow, and Member of Academia Europaea.

Ping Zhang (Fellow, IEEE) is currently a Full Professor and Director of the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. He is an Academician with the Chinese Academy of Engineering (CAE). He is also a member of the IMT-2020 (5G) Experts Panel and the Experts Panel for China's 6G development.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on November 15,2023 at 07:16:35 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.