Rendering Multi-party Mobile Augmented Reality From Edge

Lei Zhang*†, Andy Sun†, Ryan Shea†, Jiangchuan Liu†, Miao Zhang†
{lza70,hpsun,rws1,jcliu,mza94}@sfu.ca
*College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China
†School of Computing Science, Simon Fraser University, Burnaby, Canada

ABSTRACT

Mobile augmented reality (MAR) augments a real-world environment (probably surrounding or close to the mobile user) by computer-generated perceptual information. Utilizing the emerging edge computing paradigm in MAR systems can reduce the power consumption and computation load for the mobile devices and improve responsiveness of the MAR service. Different from existing studies that mainly explored how to better enable the MAR services utilizing edge computing resources, our focus is to optimize the video generation stage of the edge-based MAR services–efficiently using the available edge computing resources to render and encode the augmented reality as video streams to the mobile clients. Specifically, for multi-party AR applications, we identify the advantages and disadvantages of two encoding schemes, namely colocated encoding and spilt encoding, and examine the trade-off between performance and scalability when the rendering and encoding tasks are colocated or split. Towards optimally placing AR video rendering and encoding in the edge, we formulate and solve the rendering and encoding task assignment problem for multi-party edge-based MAR services to maximize the QoS for the users and the edge computing efficiency. The proposed task assignment scheme is proved to be superior through extensive trace-driven simulations and experiments on our prototype system.

CCS CONCEPTS

• **Information systems** → **Multimedia content creation**; • **Computing methodologies** → **Mixed / augmented reality**; • **Networks** → *Cloud computing*; • **Human-centered computing** → *Ubiquitous and mobile computing systems and tools.*

KEYWORDS

Augmented Reality, Mobile, Edge Computing

ACM Reference Format:
Lei Zhang*†, Andy Sun†, Ryan Shea†, Jiangchuan Liu†, Miao Zhang†. 2019. Rendering Multi-party Mobile Augmented Reality From Edge. In *29th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '19), June 21, 2019, Amherst, MA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3304112.3325612

NOSSDAV '19, June 21, 2019, Amherst, MA, USA

ACM ISBN 978-1-4503-6298-6/19/06…$15.00
https://doi.org/10.1145/3304112.3325612

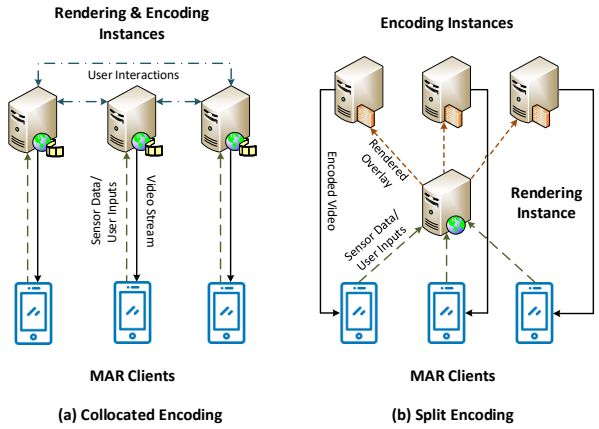

Figure 1: MAR rendering and encoding

1 INTRODUCTION

Integrating powerful sensing capability and unparalleled mobile communication abilities smartphones have led to a plethora of new and exciting applications, e.g., cognitive assistance, virtual/augmented reality (VR/AR). Such mobile augmented reality (MAR) is extremely promising for a wide range of applications such as gaming, tourism, entertainment, advertisement, education, and manufacture [2]. It has been reported that MAR will be the primary drive of a $108 billion virtual/augmented reality market by 2021[1]. In response to such great popularity and huge market increase, major industry players have released their AR develop platforms, such as Apple's ARKit[2] and Facebook's AR Studio[3], to incubate various novel MAR apps. Other than the single-device AR applications, multi-party AR applications are emerging and becoming more and more popular, which allow multiple users to share and interact with the same augmented reality and thus significantly enhance the user experience. Examples of multi-party AR applications can be the AR games[4] that allow multiple players to participate and compete with other. Another timely example is augmented vehicular reality[10], which utilizes visual information from nearby vehicles to broaden the vehicle's visual horizon through AR. Rather than being generated by each client individually, such multi-party AR is contributed and affected by all the users.

[1]https://www.digi-capital.com/news/2017/01/after-mixed-year-mobile-ar-to-drive-108-billion-vrar-market-by-2021/
[2]https://developer.apple.com/arkit/
[3]https://developers.facebook.com/products/ar-studio
[4]https://developer.apple.com/documentation/arkit/swiftshot_creating_a_game_for_augmented_reality

Supporting the resource-hungry and delay-sensitive AR applications is not an easy task for mobile devices. The amount of computation required to analyze data from the smartphone sensors, render the augmented elements, and finally compose the AR scene demands considerable power from mobile devices. As mobile devices are energy limited due to battery restrictions, a rich AR application can quickly drain even the largest smartphone batteries. Cloud-based MAR systems [1] are designed to reduce the power consumption and the computation load for mobile devices by exploiting the high-end CPUs and GPUs in powerful cloud servers. However, due to the long delay and expensive bandwidth, cloud computing becomes unable to meet the stringent requirements of latency-sensitive applications such as MAR. It is also impractical to transmit all the ever-growing distributed data over today's already-congested backbone networks to the remote cloud. The emerging mobile edge computing paradigm becomes a timely solution [12], in which highly responsive computing nodes are placed in close proximity to mobile devices. Existing studies show that edge-based solution can reduce the service response time up to 200 ms compared to the conventional cloud offloading approach [4].

Existing enhancements on MAR systems mainly explored how to better enable MAR services: tailoring the computation-intensive computer vision algorithms for executing on mobile devices [8] and reducing the amount of data required to be uploaded to remote servers [9]. Different from pervious works, our focus is to optimize the later stage, the video generation of edge-based MAR services–using available resources in the edge to render and encode the augmented reality as video streams to the users. We attempt to investigate the problem of the rendering and encoding task placement for multi-party MAR services to maximize QoS for the users and the edge computing efficiency. In MAR systems, different task types (i.e., rendering tasks and encoding tasks) can be colocated in the same edge server or split to multiple edge servers, as shown in Fig. 1. We refer to the two cases as (a) colocated encoding and (b) split encoding, respectively, indicating whether the rendering and the encoding are accomplished on the same server. Intuitively, colocated encoding could provide the best QoS (e.g., lowest rendering delay) to a single user. However, in a multi-party AR session, putting rendering and encoding tasks for all the users together may lead to severe resource shortage/contention, while keeping them sperate may increase the interaction delay between multiple users. To this end, split encoding can provide a better scalability with low interaction delay for a large number of users, but it may impair single-user QoS inevitably due to the increased system complexity and the introduced communication overhead.

The less-studied problem is challenging since it is difficult to mitigate the trade-off between the MAR service performance and the edge system's scalability. In this paper, we attempt to solve the featured and compelling rendering/encoding task placement problem for edge-based multi-party MAR systems. We first identify the advantages and disadvantages of two encoding schemes in the case of multi-party MAR systems. We next propose the resource demand models and the performance metric models, and formulate the multi-party AR video rendering/encoding task placement problem. The optimization problem is solved by carefully designed heuristic algorithms. Finally we implement and evaluate the performance of a prototype edge-based multi-party MAR system.

2 BACKGROUND AND MOTIVATION

Edge-based MAR systems are offloaded systems, which consist of mobile client side and edge server side. Figure 2 illustrates a high-level breakdown of the system framework. The mobile client collects and transmits AR metadata and user inputs to the edge cloudlets that is responsible for remotely rendering and encoding the AR video following the application logic. Rather than extracting information from a user's surroundings for MAR services, our focus in this work is to generate the augmented video scenes, which is a critical and resource-demanding task.

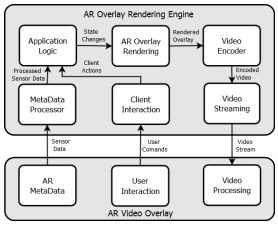

Figure 2: Edge-based MAR framework

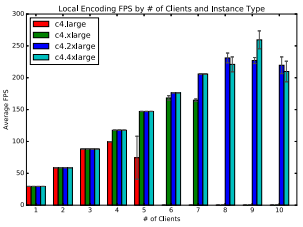

Figure 3: Colocated encoding performance

Naively, one may expect that a powerful edge server with a high-end GPU can easily support the rendering and encoding for a large number of MAR clients. However, this is not true in practice. The traditional use-case for a GPU is rendering a graphic to a monitor, a highly optimized rendering pipeline where frames are asynchronously rendered in parallel and sent to the framebuffer and from there to the monitor. In case of edge-based MAR services, since the output target is a mobile thin client wirelessly connected to the server, the contents of the framebuffer must be read after every frame, breaking the rendering pipeline and forcing the GPU to wait until the current frame is finished before beginning the next, which leads to a much less efficient rendering process and thus higher consumption for resources.

2.1 Colocated *vs* Split Encoding

A straight-forward way of generating the AR video is to render and encode the video steam at the same server, which is referred to as the colocated encoding scheme. The colocated encoding introduces the least overhead, while it may not be scalable for supporting multiple clients due to the mentioned inefficient resource usage. We next examine the scalability of colocated encoding. We virtualize the colocated encoding instances on our university's computing server clusters with 4 different configurations that attempt to mirror Amazon Web Services' (AWS) Elastic Compute Cloud's C4 offerings[5]. Each configuration doubles the core count and memory of the lower one. We deploy a single server instance on each of the 4 configurations to support different number of clients. The server is implemented in NodeJS and uses FFMPEG to encode the video streams on the CPU. We measure the aggregate frames per second (FPS) output over 10 second intervals, and plot the results in Figure 3.

The lack of a result for a given client count on a instance type in Figure 3 is generally due to the server being unable to keep

[5]https://aws.amazon.com/ec2/instance-types/

Clients	CPU% (ffmpeg+node)	GPU%	Tx (kB/s)
1	34.17+7.27	1.64	107.77
2	83.86+12.57	3.21	214.18
3	143.82+18.17	4.96	321.93
4	218.52+23.07	6.62	422.37

Table 1: Colocated Encoding Profiling

up with the number of clients and resulted in taking too long to render the frames in an acceptable time span. An exception is that the *c4.large* instance ran out of memory as it attempted to buffer the data to be piped into the encoders for any client count greater than 4. Figure 3 evidences the unacceptable (vertical) scalability of colocated encoding, since it could only support 2-3 additional clients per doubling of core count and memory. Even with the standardized commercial cloud servers, colocated encoding can only support a very limited number of users by one instance, not to mention using the less-powerful edge servers. To further identify the bottleneck of resources, we profile three basic metrics, CPU utilization, GPU utilization, and network transmissions, averaged over a period of 90 seconds while varying the number of clients to roughly determine the cause of inefficiencies. Table 1 shows that the resource contention is mainly from CPU and the major cause is video encoding, the heavy CPU-based task from FFMPEG, whose CPU utilization increases nonlinearly with the number of concurrent clients.

In multi-party MAR services, users can share and interact with the same AR world. According the MAR system framework, the rendering server executes the application logic before rendering the scene. If the video steams for the participants of a multi-party AR session are rendered separately at different servers, the application logics are then executed remotely from each other. In such case, whoever initiates a change in the multi-party AR world, the application logic needs to be updated and synchronized to all the participants at geo-distributed rendering servers, which may implies long interaction delay. Therefore, putting the rendering for those users of a multi-party AR session on the same server (physical machine) or the least number servers is important for synchronizing the video streams to minimize the interaction delay and thus provide the uniform user experience.

Apparently, the colocated encoding cannot work well in this situation due to its limited scalability. Since the video encoder is a black box, the encoding process can be offloaded to separate edge cloudlets to remove the computation bottleneck, which is referred to as the split encoding scheme. Although the split encoding is a practical approach to better scale out the MAR systems, it is not cost-free. First, offloading the video encoders adds extra communication hops into the MAR systems, which increases the service latency by the RTT between the rendering and encoding servers. As the key performance metric for such latency-sensitive applications, it undoubtedly hurts user experience. Second, split encoding introduces extra operation costs, including the activation cost for opening/running the encoding servers and the bandwidth cost for the data transmissions between the rendering servers and the encoding servers. Given the heterogeneity of the edge resources and the user demands for MAR services, the assignment of rendering and encoding tasks should be carefully considered. A good rendering/encoding task assignment scheme should be flexible to choose from colocated encoding and split encoding to guarantee the satisfactory QoS for multiple users, and be able to optimally patch the rendering and encoding tasks so that the limited edge resources can be efficiently utilized.

3 SYSTEM MODEL

We consider a mobile edge network with n MAR clients and m heterogeneous edge servers. Denote $U = \{u_1, u_2, ..., u_n\}$ as the set of MAR clients and $S = \{s_1, s_2, ..., s_m\}$ as the set of edge servers. The delay between any client and server (any two servers) is given by the function $d(u, s)$ for $u \in U, s \in S$ ($d(s, s')$ for $s, s' \in S$). We set $x^R_{u,s}$ as the binary decision variable to denote whether the rendering task from client u is placed on server s, set $x^E_{u,s}$ as the decision variable for the encoding task, and set $x^{RE}_{u,s}$ as the decision variable for the colocated rendering and encoding task. In addition, assume that the users are divided into k different groups $U_1, U_2, .., U_k$. We use a user group to indicate the users/clients in the same multi-party AR session as mentioned in the previous section, and thus the rendering tasks from the same user group should be assigned to the same/least number of servers.

3.1 Characterizing Resource Demands

We first propose the the analytical model to abstract the complicated features of the resource demands. Typically, a task's resource demand is affected by 3 factors: the task type t, the task complexity c, and the influence from the assigned server f. Therefore, we adopt the following model

$$Computation\_Demand(t, c, f) = c_t(r_u) \cdot f_t(N_s), \quad (1)$$

where the task type $t \in T = \{R, E, RE\}$, r_u denotes the resolution (the number of pixels in one frame) of the video streamed to client u, and $N_s = \sum_{u \in U}(x^R_{u,s} + x^E_{u,s} + x^{RE}_{u,s})$ is the number of tasks assigned to server s. In the proposed analytical model, $c_t(r_u)$ and $f_t(N_s)$ are fitted functions for the task complexity and influence factor from the server, respectively, using the data acquired from the measurements in the last section. Rather than getting the exact mathematical results for the test system, the goal of fitting this model is to characterize the resource demands for rendering and encoding tasks that can be easily re-captured by measuring other MAR systems.

3.2 Performance Metric Models

3.2.1 User Group Rendering Locality. Since the users in the same user group (in one multi-party AR session) can interact with same AR world and thus affect others' application logic, if their renders are located on the same edge server, the interaction delay between multiple parties in the AR session can be minimized, which can help provide uniform user experience for MAR services. Therefore, it is crucial to put together the video rendering for the participants of a multi-party AR session. For each user group U_i, the number of edge servers that are used to render for the users in this group can be calculated as $\sum_{s \in S} \max_{u \in U_i}(x^{RE}_{u,s}, x^R_{u,s})$. We define the locality metric as the maximum number of assigned servers for rendering for one user group

$$L = \max_{i \in [1,k]} \sum_{s \in S} \max_{u \in U_i}(x^{RE}_{u,s}, x^R_{u,s}). \quad (2)$$

We want to maximize the user group rendering locality, because higher rendering locality implies lower interaction delay for the user group. Given the above definition, maximizing the user group rendering locality is to minimize L and thus minimize the QoS variance within the same user group.

3.2.2 Transmission Latency. The service latency is one of the most important metric to evaluate edge-based MAR systems. For colocated encoding, the round-trip latency between the server to the client is $2\sum_{s\in S} x^{RE}_{u,s} d(u,s)$. For split encoding, the latency consists of three parts, from the client to the renderer, from the renderer to the encoder and from the encoder to the client, $\sum_{s,s'\in S\wedge s\neq s'}(x^R_{u,s}d(u,s)+ x^R_{u,s}x^E_{u,s'}d(s,s') + x^E_{u,s'}d(u,s'))$. In the network edge, $d(u,s)$ and $d(s,s')$ should be easy to measure or estimate, since either of the communications should have very few hops. The total transmission latency for one user group (one AR session) can be given as

$$D = \sum_{u\in U}[2\sum_{s\in S} x^{RE}_{u,s} d(u,s)+ \sum_{s,s'\in S\wedge s\neq s'}(x^R_{u,s}d(u,s) + x^R_{u,s}x^E_{u,s'}d(s,s') + x^E_{u,s'}d(u,s'))]. \quad (3)$$

The transmission latency here does not include the task execution time, since it is not affected by the selection of servers when enough computation resource is provisioned. We calculate D for each user group, which is also the transmission latency for the corresponding multi-party AR session.

3.2.3 Server Operational Cost. For each of the edge servers, if there is at least one rendering/encoding task assigned to it, a one-time static cost has to be paid for activating the server. Such operational cost is related to the capacity/configuration of the server/instance and irrespective of the number of tasks/clients or the type of the workload assigned to the server. Denote e_s as the operational cost for server s. The total operational cost for all edge servers can be given as

$$E = \sum_{s\in S}\mathbb{I}[N_s > 0]e_s = \sum_{s\in S}\mathbb{I}[\sum_{u\in U}(x^R_{u,s} + x^E_{u,s} + x^{RE}_{u,s}) > 0]e_s, \quad (4)$$

where $\mathbb{I}[\cdot]$ is the indicator function.

3.2.4 AR Video Quality. The video quality is directly related to the user experience in MAR services. We use r_u, the number of pixels per frame for the video streamed to client u to represent the video resolution. The user experience can be modeled as a concave function of video resolution/bitrate $q(r_u)$ (e.g., $q(r_u) = a\log(r_u) + b$) [3]. The overall subjective QoS of the MAR system can be given as

$$Q = \sum_{u\in U} q(r_u). \quad (5)$$

3.3 Problem Formulation and Solution

3.3.1 Optimization Objective. Our goal is to find the task assignment $\mathcal{X} = \{(x^R_{u,s}, x^E_{u,s}, x^{RE}_{u,s})|u \in U, s \in S\}$ and the resolution setting $\mathcal{R} = \{r_u|u \in U\}$ that minimize the service delay and the operational cost and maximize the video quality. To formulate the multi-objective optimization problem, we adopt the weighted sum method and introduce three weighting parameters α, β and γ to reflect the preference. Therefore, the optimization problem $\mathcal{P}$ can be formulated as

Algorithm 1: Task Placement

```
Initialize R_0, α, β, and τ;
R ← R_0, i ← 0;
while true do
    X ← solve Problem P with fixed R (described in detail below);
    R ← solve Problem P with fixed X;
    F_i(X, R) = L + αD + βE − γQ;
    if |(F_i − F_{i−1})/F_i| ≤ τ then
        break;
    end
    i ← i + 1;
end
return X and R.
```

$$\textbf{Minimize } F(\mathcal{X},\mathcal{R}) = L + \alpha D + \beta E - \gamma Q; \quad (6)$$

$$\textbf{s.t.} \sum_{t\in T}\sum_{u\in U} x^t_{u,s} f_t(N_s) c_t(r_u) \le C_s, \forall s \in S; \quad (7)$$

$$\sum_{s\in S} x^R_{u,s} + x^{RE}_{u,s} = 1, \forall u \in U; \quad (8)$$

$$\sum_{s\in S} x^E_{u,s} + x^{RE}_{u,s} = 1, \forall u \in U; \quad (9)$$

$$x^R_{u,s}, x^E_{u,s}, x^{RE}_{u,s} \in \{0,1\}, \forall u \in U, \forall s \in S; \quad (10)$$

where C_s denotes the computation capacity of server s and $B(s,s')$ indicates the available bandwidth between server s and server s'. Eq. 7 ensures that the workload of the tasks assigned to an edge server does not exceed its capacity. Eq. 8 and Eq. 9 indicate each rendering and encoding task should be assigned to some server.

3.3.2 Optimization Solver. The formulated problem is a nonlinear integer programming problem, which is known to be difficult to solve [6]. One of the reasons for the problem's hardness is from the multiple decision variables and multiple conflicting objectives. To solve the optimization problem, our intuition is to divide and conquer these objectives and variables. We first develop a two-step heuristic solution generally based on the block coordinate descent method [5]. The main idea is to solve the problem $\mathcal{P}$ with fixed $\mathcal{R}$ and $\mathcal{X}$, respectively, and the decision variables are optimized iteratively until the objective function F converges to a certain range of τ. As shown in Algorithm 1 (step 1), we sequentially fix one variable, i.e., $\mathcal{R}$ and $\mathcal{X}$, and update the other one. It is worth noting that, in practice MAR services usually provide discrete resolution selections, e,g, 360p, 720p 1080p, which suggests that Algorithm 1 may only need a very limited number of iterations. The initial resolution setting $\mathcal{R}_0$ can be specified as by the user/system preferences, (e.g., the minimal resolution that is acceptable by the users). Solving $\mathcal{P}$ with fixed $\mathcal{X}$ is relatively simple, since the special case of the problem is convex with respect to $\mathcal{R}$. However, solving $\mathcal{P}$ with fixed $\mathcal{R}$ is still hard, at least as hard as multiple Knapsack problem.

Following the intuition of divide and conquer, we propose another heuristic (step 2) to solve $\mathcal{P}$ with fixed $\mathcal{R}$: (1) We first check if there is a server with enough capacity that can accommodate $f_{RE}(|U_i|)c_{RE}(r)$, the computation demands for the colocated tasks. (2) If so, we search all the qualified servers and assign the colocated tasks to the one producing the minimal objective value; (3) If not, split encoding is selected and we employ a similar procedure to assign the rendering tasks and the encoding tasks. The only difference is that, for the colocated tasks and the rendering tasks we search and assign the server for the whole user group, while for the encoding tasks we check the servers for each user individually. The time complexity for this heuristic is bounded by the placement of the encoding tasks, which is $O(k|U_i|m)$. In the worst case, we employ split encoding for all the user groups, and thus need to place the rendering and encoding tasks for every user, where the time complexity is $O(nm)$.

4 PERFORMANCE EVALUATION

4.1 MAR System Implementation

We design a test AR application to render an avatar of a virtual object (e.g., the earth in Figure 4) and attach it to the client's camera stream at a specific location and pose. With the assistance of the sensors, we can estimate the camera pose by reading the orientation from the gyroscope and position from the GPS, which can be easily translated into XYZ coordinates in a virtual world. To enable the multi-party interaction, we set the application logic to be that whenever the avatar is touched by a user on the screen its color is changed, which is light-weight and easy to be synchronized.

We adopt the system framework in Figure 2. The server was implemented in NodeJS primarily due to the abundance of pre-existing support libraries for Three.js, a popular open-source API for manipulating 3D graphics, along with the versatile offered functionality. The *MetaData Processor* (MDP) and *Client Interaction* modules ingest, validate, and process the client data by performing sensor fusion to predict and reduce noise from the incoming sensor data. Based on the received data, *Application Logic* computes the updates to the virtual world, which is passed to the *AR Overlay Rendering* module. The generated image frames are then sent to the *Video Encoder*. The black box encoder converts the raw image data into an alpha-channel enabled VP8-encoded WebM bytestream that is then forwarded to client. The client is a simple HTML webpage that utilizes the device's underlying sensors and camera to determine the user's pose as well as displaying the surrounding physical environment. It establishes a connection with the server using a WebSocket; sending the server orientation updates while receiving the corresponding encoded video stream. This stream is then overlayed on top of the camera to provide a pixel-perfect alignment between the real and virtual objects.

To realize the split encoding scheme, we break the rendering pipeline to refactor the encoder so that it can be offloaded to a separate server. Offloading the encoder may cause a bandwidth issue, as raw images/frames need to be transmitted from the render to the encoder. To solve the problem, we cannot perform software encoding as we would otherwise return to square one; a hardware-accelerated encoder would be a viable alternative, but there are currently no encoding formats that officially support alpha channels. To this end, we add in an LZ4 compression module prior to sending the image data to the encoder, whose lossless compression can perform at line-rate. This algorithm takes advantage of that our data, when viewed as a scanline bytestream, is sparse and highly repetitive: a large portion of the AR video overlay will be transparent resulting in a repeated 4-byte pattern of (0,0,0,255) until a virtual object is present.

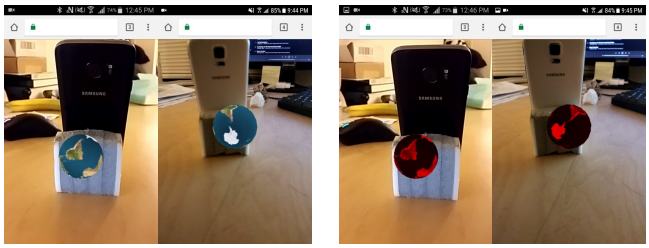

Figure 4: A snapshot of the prototype edge-based MAR system with multi-party interaction (before/after)

4.2 Evaluation Results

We simulate the user demand based on the two-week long Pokemon Go dataset [11], consisting of Pokemon Gym EXP changes at different locations. We take every 1000 EXP change as a user joining a AR session for 10 minutes. The users at the same location are assumed to join one multi-party AR sessions, and thus can be classified into 12 groups (12 multi-party AR sessioins). Figure 5 plots the average hourly user demand per user group, which suggests peak hours are often at night and weekdays have larger variances than weekends. Our prototype system is deployed to our university's local computing research platform, backed by 9 TB of RAM and 1000 logical CPUs split across three physical racks with 10 Gb/s Ethernet switches. This server instance is provisioned with 4x2.4 GHz cores, 16 GB of RAM, and an NVIDIA GRID-K1 GPU with the encoding done using FFMPEG on the CPU. The client is a Samsung Galaxy S7 (SGS7) on Android 6.0.1 running Google Chrome 55.0.2883.91. We introduce a random transmission delay from 0 to 50 ms following a uniform distribution [7] to emulate network jitters. For comparison, we implement three other task placement algorithms to minimize the transmission latency (Min_Latency), minimize the system operating cost (Min_Cost), and maximize the AR video quality (Max_Resolution), respectively. We adopt three performance metrics – average transmission latency, hourly operating monetary and AR video resolution.

We first check the effects of varying weights. We fixed $\alpha = 1$ and changed the values of β and γ to examine the tradeoffs between the conflict goals. Figure 6 shows the transmission latencies and the hourly operating costs of the MAR system when the ratio of β/α change from 0 to infinity. Figure 7 plots the same performance metrics when we vary the ratio of γ/α. From the figures, we can see clear tradeoff between the transmission latency and the operating cost. We also observe that the operating cost increases when we weight more on the video quality, while the transmission latency does not show the same trend (and thus ignored in Figure 7). Based on the results, we adopt the optimal weight setting as $\alpha = 1, \beta = 100, \gamma = 200$.

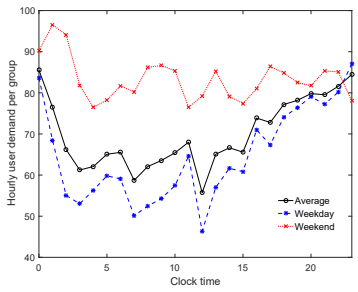

Figure 5: User demand

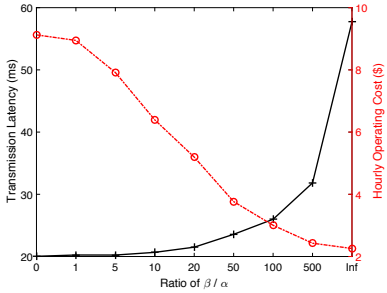

Figure 6: Effects of α, β

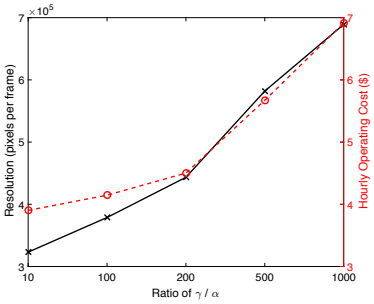

Figure 7: Effects of α, γ

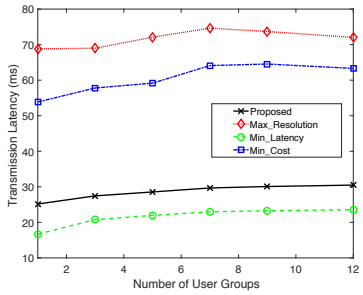

Figure 8: Transmission latency *vs* Num of user groups

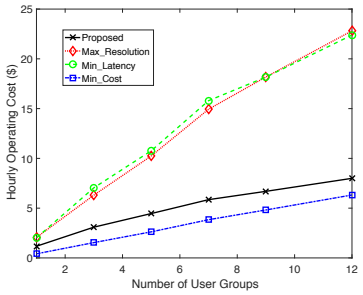

Figure 9: Monetary cost *vs* Num of user groups

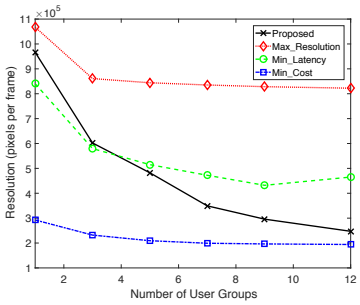

Figure 10: Video resolution *vs* Num of user groups

We next examine the system scalability when handling the user demands from different number of user groups. As one user group is considered to have a multi-party AR session, varying the number of user groups means the MAR system is required to support different number of concurrent multi-party AR sessions. As shown in Figure 8, our approach has close performance to Min_Latency (much better than Min_Cost and Max_Resolution) in terms of the average per-user transmission latency, which does not change much against different user demand. When the user demand grows, Min_Latency and Max_Resolution have significant operating cost increases, while our proposed approach keeps relatively low operating cost as shown in Figure 9. However, Figure 10 shows that the proposed approach trades the video quality to achieve good performance in transmission latency and operating cost, which can be changed by increasing the value of γ.

5 CONCLUSIONS

In this work, we systematically studied the AR video rendering and encoding task placement in the edge-based MAR system. We identified the trade-off between scalability and performance using two encoding schemes, namely colocated encoding and split encoding, for the multi-party AR applications. We formulated and solved the rendering/encoding task placement optimization problem. A prototype edge-based MAR system is further implemented, and our rendering/encoding task placement scheme is proved to be superior through evaluations.

REFERENCES

[1] Dimitris Chatzopoulos, Carlos Bermejo, Zhanpeng Huang, Arailym Butabayeva, Rui Zheng, Morteza Golkarifard, and Pan Hui. 2017. Hyperion: a wearable augmented reality system for text extraction and manipulation in the air. In *Proceedings of ACM MMSys 2017*. ACM, 284–295.

[2] Dimitris Chatzopoulos, Carlos Bermejo, Zhanpeng Huang, and Pan Hui. 2017. Mobile augmented reality survey: From where we are to where we go. *IEEE Access* 5 (2017), 6917–6950.

[3] Chao Chen, Xiaoqing Zhu, Gustavo de Veciana, Alan C Bovik, and Robert W Heath. 2015. Rate adaptation and admission control for video transmission with subjective quality constraints. *IEEE JSTSP* 9, 1 (2015), 22–36.

[4] Zhuo Chen, Wenlu Hu, Junjue Wang, Siyan Zhao, Brandon Amos, Guanhang Wu, Kiryong Ha, Khalid Elgazzar, Padmanabhan Pillai, Roberta Klatzky, et al. 2017. An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 14.

[5] Luigi Grippo and Marco Sciandrone. 2000. On the convergence of the block nonlinear Gauss–Seidel method under convex constraints. *Operations research letters* 26, 3 (2000), 127–136.

[6] Raymond Hemmecke, Matthias Köppe, Jon Lee, and Robert Weismantel. 2010. Nonlinear integer programming. In *50 Years of Integer Programming 1958-2008*. Springer, 561–618.

[7] Wenlu Hu, Ying Gao, Kiryong Ha, Junjue Wang, Brandon Amos, Zhuo Chen, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2016. Quantifying the impact of edge computing on mobile applications. In *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems*. ACM, 5.

[8] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. 2017. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of ACM MobiSys 2017*. ACM, 82–95.

[9] Puneet Jain, Justin Manweiler, and Romit Roy Choudhury. 2016. Low Bandwidth Offload for Mobile AR. In *Proceedings of ACM CoNEXT 2016*. ACM, 237–251.

[10] Hang Qiu, Fawad Ahmad, Fan Bai, Marco Gruteser, and Ramesh Govindan. 2018. AVR: Augmented Vehicular Reality. In *Proceedings of ACM MobiSys 2018*. ACM, 81–95.

[11] Ryan Shea, Di Fu, Andy Sun, Chao Cai, Xiaoqiang Ma, Xiaoyi Fan, Wei Gong, and Jiangchuan Liu. 2017. Location-based augmented reality with pervasive smartphone sensors: Inside and beyond pokemon go! *IEEE Access* 5 (2017), 9619–9631.

[12] Lin Wang, Lei Jiao, Ting He, Jun Li, and Max Mühlhäuser. 2018. Service entity placement for social virtual reality applications in edge computing. In *Proceedings of INFOCOM*.