DARE: Dynamic Adaptive Mobile Augmented Reality with Edge Computing

Qiang Liu and Tao Han
Electrical and Computer Engineering Department,
The University of North Carolina at Charlotte, NC, United States
Email: {qliu12, tao.han}@uncc.edu

Abstract—Mobile augmented reality (MAR) is a killer application of mobile edge computing because of its high computation demand and stringent latency requirement. Since edge networks and computing resources are highly dynamic, handling such dynamics is essential for providing high-quality MAR services. In this paper, we design a new network protocol named DARE (dynamic adaptive AR over the edge) that enables mobile users to dynamically change their AR configurations according to wireless channel conditions and computation workloads in edge servers. The dynamic configuration adaptations reduce the service latency of MAR users and maximize the quality of augmentation (QoA) under varying network conditions and computation workloads. Considering the video frame size and computation model, i.e., object detection algorithms, as two key parameters in adapting the AR configuration, we develop analytical models to characterize the impact of these parameters on QoA and the service latency. Then, we design optimization mechanisms on both the edge server and AR devices to guide the AR configuration adaptation and server computation resource allocation. The performance of the DARE protocol is validated through a small-scale testbed implementation.

Index Terms—Edge computing; Mobile augmented reality; Adaptive computing

I. INTRODUCTION

Augmented reality (AR) is able to embed virtual information seamlessly into our physical environment and provide new kinds of experiences where the world is improved by virtual content blending with the real [1]. Many market forecasts are expecting a widely adoption of AR in various industries such as tourism, entertainment, advertisement, education, manufacture and so on [2], [3]. The key component of AR is a fast and precise detection and understanding of physical environment so that virtual contents can synchronize with the real world.

The environmental understanding is enabled by image sensors integrated with AR devices. The image data from the sensors are processed by computer vision algorithms to detect objects in the physical environment. However, the computation complexity of computer vision algorithms, e.g., SSD [4], YOLO [5], and Faster R-CNN [6], are usually too high to run on mobile devices. Although there are some low-complexity object detection frameworks and algorithms such as Google MobileNet [7], Tiny YOLO [5] and DeepMon [8] that can perform object detection on mobile devices, the performance of these algorithms in terms of the mean average precision (mAP)\(^1\) and latency are significantly worse than that of the state-of-the-art computer vision algorithms. In addition, because of the hardware limitation, a mobile device cannot perform advanced computer vision analysis, e.g., human action detection [10], which limits the virtual information that can be generated from the physical world.

Mobile edge computing (MEC) is a promising paradigm to bridge the gap between the stringent computation demand of AR and the constrained computation resources on mobile devices [11]. As illustrated in Fig. 1, with MEC, mobile AR (MAR) clients offload image data to an edge server via wireless access point while the edge server performs environmental understanding, i.e., object detection, and sends the corresponding virtual content back to MAR clients. Edge servers are provisioned with sufficient computation capacity to perform advanced computer vision analytics. Since edge servers are placed in close proximity to users, the communication link between the wireless access point and edge server usually has a very low latency in order to provide highly responsive services. However, wireless links between MAR clients and the wireless access point are highly dynamic and may introduce a long latency during the data offloading and virtual content delivery. Besides, edge servers are usually less powerful than cloud servers and thus can be easily overloaded by a number of MAR clients. Therefore, handling the dynamic wireless links and limited computation resources is crucial to the edge-based MAR system.

In this paper, we design the DARE (dynamic adaptive AR over the edge) protocol that enables the edge-based MAR system to dynamically change AR configurations and computation resource allocations according to wireless channel conditions and available computation resources on the edge server. In the adaption, the DARE protocol trades off the quality of augmentation (QoA) against the service latency. The intuitive basis of DARE is that performing precise AR

\(^1\)The mean average precision (mAP) is a metric to evaluate the detection accuracy of a visual object detection algorithm [9].
(high QoA) requires the massive data offloading and intensive computing while allowing selective approximation (medium QoA) can provide disproportionate gains in efficiency. Note that the quality of augmentation (QoA) is defined to evaluate the average object detection accuracy of the MAR service. Fig. 2 shows an example of the precise and approximate AR. The precise AR requires the computer vision algorithm to detect all objects in the image while the approximate AR only detects the objects close to the view of the AR client. There are two reasons why the precise AR introduces longer latency than the approximate AR. The first one is that the precise AR usually asks for high-fidelity image data. Transmitting the high-fidelity image data incurs the extra latency. The second one is that the precise AR often requires sophisticated computer vision algorithms that have high computational complexity.

![Precise AR](a) Precise AR. ![Approximate AR](b) Approximate AR: Low Resolution.

Figure 2: Precise vs Approximate AR.

Fig. 3 shows the overview of the DARE protocol. In the first step, MAR clients send their service requests and measurements of wireless channel conditions and service latency to a edge server. In the second step, upon receiving the requests and measurements, the optimization engine on the edge server determines the video frame sizes of MAR clients, selects computation models, which are object detection algorithms such as YOLO and SSD [4], [5], to serve different MAR clients, and optimizes the computation resource allocation by mapping the computation requests of individual MAR clients to GPUs. The video frame sizes determined by the edge server is sent back to corresponding MAR clients as AR configuration messages. In the third step, MAR clients resize their video frame sizes according to the AR configuration messages. Meanwhile, MAR clients also adapt their video frame rate based on the service latency. After the frame resizing and frame rate adaptation, video frames are send to the edge server for processing. In the fourth step, the edge server sends back the virtual content that augments the reality.

With the DARE protocol, the MAR clients can adapt their frame rates and video frame sizes. The edge server can change its computation models and resource allocations for different clients. Therefore, on designing the DARE protocol, we need to optimize four parameters in the edge-based MAR system: frame rate, video frame size, computation model selection, and computation resource allocation. Since the video frame sizes are closely related to the computation model and resource allocation, we design an optimization engine on the edge server to jointly optimize the video frame size, computation model selection, and computation resource allocation. The optimal video frame sizes are fed back to the corresponding MAR clients through the AR configuration message. The MAR clients are responsible for optimizing its own video frame rate.

The technical challenges for designing the DARE protocol are as follows: 1) there are no analytical models for characterizing the impact of the image data size and computation model on QoA and the service latency for a multiuser MAR system. 2) Since the edge server is shared by multiple MAR clients, individual clients’ AR configurations are coupled with the computation resource allocation on the edge server. Such a coupling makes it computationally hard to optimally allocate computation resources and adapt clients’ AR configurations.

To address these challenges, we perform experiments to study the tradeoffs between QoA and the service latency in a MAR system. Based on the experiment results, we model QoA and the service latency as functions of the video frame size and computation model. We then formulate the AR reconfiguration and computation resource allocation in the edge-based MAR system as an optimization problem that aims to maximize the average QoA while satisfying all clients’ latency requirements. We solve the problem using the cyclic block coordinate gradient projection method and implement this solution on the edge server. On the client side, we design and implement a frame rate adaptation mechanism which adapts the number of AR video frames sent to the edge server per second according to workloads on the edge server. This mechanism helps the client to maximize the amount of virtual content obtained from the edge server. We implement the DARE protocol and evaluate its performance through experiments. The contributions of this paper are

- We designed and implemented the DARE protocol for MAR services. This protocol enables MAR clients to dynamically adapt the AR performance according to wireless channel conditions and server workloads.
- We identified the impact of the video frame size and computation model on the quality of augmentation and service latency in a multiuser MAR system and developed analytical models for studying the AR reconfiguration and computation resource allocation in the system.
- We developed optimization mechanisms on both the edge server and MAR devices to guide the AR performance adaptation and maximize the quality of augmentation while satisfying clients’ latency requirements.
To our best knowledge, we are the first to develop an adaptive MAR protocol that enables the edge-based MAR system to adapt its performance according to wireless channel conditions and available computation resources on the edge server.

II. RELATED WORK

Our work relates to mobile augmented reality, mobile cloud gaming and mobile edge computing.

Mobile Augmented Reality: Since mobile devices are constrained by their computation resources and battery power supplies, most of existing mobile augmented reality systems are developed by using either cloud or edge servers [12]–[15]. To improve the accuracy and latency of the object recognition, Jain et al. [16] designed an AR system which utilizes a location-free geometric representation of environments to prune down the visual search space. Jain et al. [17] proposed a low bandwidth offloading scheme for MAR. This scheme reduces the network latency by only transmitting the distinctive features of images to the server. Chen et al. [18] designed Glimpse which is a continuous real-time object recognition system. In this system, the authors developed an active cache mechanism to improve the recognition accuracy and proposed a trigger frame selection method to hide the transmission latency caused by wireless networks. None of the existing work on MAR dynamically changes the AR configurations and edge computation resource allocations according to wireless channel conditions and workloads on the edge servers.

Mobile Cloud Gaming: In mobile cloud gaming, mobile clients send game inputs to a remote server that executes and renders the video game. The output frames of the execution and rendering are sent back to and displayed on mobile clients [19], [20]. Since mobile cloud gaming is an interactive application, reducing the network latency of the computation offloading is the main goal in designing mobile cloud gaming solutions. Lee et al. [19] designed a speculative execution system named Outatime to hide the network latency in mobile cloud gaming. The main idea of Outatime is to render speculative frames of future outcomes and deliver the results to the mobile client one round-trip-time ahead of the next input. This solution does not apply to the edge-based MAR system because MAR interacts with the physical world that is not predefined as a video game. In a video game, since the game setting is given, it is possible to obtain highly accurate predictions on future outcomes. However, it is impossible to use the same method to predict what will appear in the view of a MAR client in the next moment. Cuervo et al. [20] proposed a collaborative rendering system called Kahawai that reduces the requirement of network bandwidth from the server to the mobile client. Kahawai uses mobile GPU to render either reduced detail or a subset of frames. In this way, Kahawai reduces the amount of data downloaded from the server and thus alleviates the bandwidth requirement. Kahawai relies on the H.264 decoder and encoder to perform the collaborative rendering. In a MAR application, the major computation is from computer vision algorithms rather than the frame rendering. Moreover, the performance of an edge-based MAR system hinges on both the uplink and downlink network bandwidth and latency. Therefore, the Kahawai solution is not appropriate for the edge-based MAR system.

Mobile Edge Computing: Edge computing has a great potential in providing the low latency computation to mobile devices [21], [22]. Therefore, exploiting edge computing to support MAR service is a promising solution. There are many works on designing edge computing systems. Satyanarayanan et al. [23] proposed the concept of the cloudlet which acts as a middle tier between the mobile devices and the data center to provide high responsive services. Yi et al. [24] designed a video analytics system on top of edge computing platform. This system enables the computation offloading from mobile users to edge nodes and exploits the collaboration among edge nodes to reduce the latency of video analytics. While these works improve the performance of edge computing system, none of them exploits the characteristics of MAR services in the system design and optimization.

III. TRADEOFFS IN EDGE-BASED MAR SYSTEM

In this section, we perform experiments to characterize the impact of video frame sizes and computation models, i.e., object detection algorithms, on QoA and the service latency for a multiuser MAR system. These experiment results not only provide the basis of modelling the adaptive MAR system but also make the case for it. Note that this paper focuses on the MAR application in which the MAR client captures the environment information via cameras and sends the information to a server for the object detection. The concept of dynamic adaptive AR can be generalized to other types of AR applications such as the surface detection and rendering.

A. Video Frame Size v.s. QoA v.s. Latency

In this experiment, we implement YOLO 544x544 [5], which is the YOLO algorithm tuned at the 544x544 image resolution, as the object detection algorithm on a workstation with an NVIDIA Quadro M4000 GPU for the computation acceleration. A MAR client is emulated in a laptop which connects to the workstation via a wireless router. We use VOC 2007 test dataset [9] for the study. To evaluate the impact of video frame sizes on QoA and the service latency, we preprocess the images in the dataset with five compression factors. For instance, when the compression factor is 0.2, an image with 500x500 pixels is resized to an image with 100x100 pixels.

Fig. 4 shows the impact of image data sizes on QoA and the service latency. As expected, a large video frame size leads to a high mAP but a long transmission latency. The gain on mAP becomes smaller as the increase of image data sizes. However, the transmission latency increases much faster with a larger video frame size. When the compression factor decreases from 1.0 to 0.6, the transmission latency decrease about 61% while
Computation latency v.s. QoA v.s. Latency

B. Computation Model v.s. QoA v.s. Latency

In this experiment, we implement four object detection algorithms based on the YOLO framework [5] and two object detection algorithms based on the SSD framework [4] on the edge server. We use the VOC 2007 test dataset [9] to evaluate mAP of the object detection and the corresponding computation latency. Fig. 6 shows the mAP and the computation latency of these algorithms in a single user MAR system. The computation model that produces the mAP. Since QoA of the computation model, QoA of the MAR client closely relates to the video frame size and the computation resource allocation on the edge server as an optimization problem, and propose an adaptive MAR algorithm to solve the problem. On modeling the system, we consider multiple MAR clients and one edge server. The MAR clients are connected to the edge server via a wireless access point.

A. System Model

Quality of Augmentation (QoA): Denote the $d_k$, $l_k$, $Q_k$, and $K$ as the image compression factor, mAP of the computation model, QoA of the $k$th MAR client, and the set of MAR clients, respectively. Here, we use the mAP value to represent a computation model. That is, given a mAP, we find a MAR client closely relates to the video frame size and the computation model, $Q_k$ can be expressed as $Q_k = Q(d_k, l_k)$.
where \( Q(x, y) \) is a function of \( x \) and \( y \). Hence, the average QoA of MAR clients in the system is modeled as

\[
Q = \frac{1}{|K|} \sum_{k \in K} Q_k.
\]

(1)

Although the computation models have the frame resolution embedded, e.g., YOLO 544x544, we consider the computation model, i.e., mAP, and frame resolution, i.e., image compression factor, as two separate parameters for two reasons. First, given a computation model, the compression factor affects QoA of MAR services as shown in Fig. 4. Second, the number of computation models that can be supported in a practical system is limited by the computation resources. Therefore, the corresponding mAPs are discrete. However, the compression factor is continuous and selected by MAR clients according to the system performance.

**Service Latency:** We model the service latency as the static latency and dynamic latency. The static latency does not change when we adapt the configurations of MAR. For example, the latency of the communication link establishment and image preprocessing are considered as the static latency in a MAR system. The dynamic latency depends on the image data size and computation model of a MAR system. In our system, we consider the transmission latency in transmitting image data and computation latency in performing the object detection as the dynamic latency. Hence, the service latency of the \( k \)th MAR client is expressed as

\[
T_k = T_k^s + T_k^p + T_k^c
\]

(2)

where \( T_k^s \) is the transmission latency, \( T_k^p \) is the computation latency, and \( T_k^c \) is the static latency in the system. In a practical system, the transmission latency consists of the wireless transmission latency between mobile clients and wireless access points and the wired transmission latency between wireless access points to edge servers. Since the wireless transmission latency is expected to be much longer and more dynamic than the wired transmission latency, we focus on the impact of the wireless transmission latency on the QoA of MAR services. Denote \( r_k \) as the wireless data rate of the \( k \)th MAR client. Let \( S(d_k) \), which is a non-decreasing function with respect to \( d_k \), be the image data size of the \( k \)th client. Then, \( T_k^s = S(d_k)/r_k \).

The computation latency depends on the computational complexity of the computation model and available computation resources on the edge server [25]. Let \( c_k \) be the computational complexity of the \( k \)th client’s computation model. We define \( f_k \) as the computation resources allocated to the \( k \)th client by the edge server. Then, the computation latency experienced by the \( k \)th client can be modeled as \( T_k^p = P(l_k)/f_k \). Here, \( P(l_k) \), which is a function of \( l_k \), represents the computational complexity of the client’s request.

**B. Problem Formulation**

On designing the optimization engine, we aim to maximize the average QoA while satisfying the latency requirements of the MAR clients in the system. The variables are \( d_k \), \( l_k \), and \( f_k \), \( \forall k \in K \). The optimization problem can be formulated as

\[
\mathcal{P}_0 : \quad \max_{\{d_k, l_k, f_k, \forall k \in K\}} Q \quad s.t. \\
C_1 : \quad T_k \leq T_k^{max}, \forall k \in K; \\
C_2 : \quad \sum_{k \in K} f_k \leq F^{max}; \\
C_3 : \quad d_k^{min} \leq d_k \leq d_k^{max}, \forall k \in K; \\
C_4 : \quad l_k \in \{l_k^{min}, ..., l_k^{max}\}, \forall k \in K;
\]

(3)

where \( T_k^{max} \) is the maximum tolerable latency of the \( k \)th client, and \( F^{max} \) is the total computation resources on the edge server. As the emergence of the network slicing technology, it is reasonable to assume that a network slice is created for supporting a particular service, e.g., the MAR service [26]. Hence, we consider the edge server is dedicated to the MAR service and assume that the total computation resource for the MAR service is known as \( F^{max} \). The constraints \( C_1 \) guarantee that the service latency of users are not larger than their maximum tolerable latency; the constraint \( C_2 \) means that the allocated computation resources do not exceed the total computation resources on the edge server; the constraints \( C_3 \) and \( C_4 \) are the constraints of the image compression factor and mAP of the computation model.

Note that \( l_k \) is a discrete variable. The values of \( l_k \) depend on the available computation models in the system. In other words, each computation model corresponds to a value of \( l_k \). Therefore, deciding \( l_k \) equals to selecting the computation model. Hence, \( l_k \) acts as an integer variable that selects the computation models in the optimization. As shown in the experiment results in Sec. III, QoA of a MAR client is a nonlinear function of the image data size. Thus, the above optimization problem is a mixed-integer non-linear programming problem (MINLP) which is difficult to solve [27].

**C. Optimization Algorithm**

We develop an adaptive MAR optimization algorithm to efficiently solve the above problem \( \mathcal{P}_0 \). According to the solution, we determine the AR video frame compression factor and select computation models for individual clients.

To solve problem \( \mathcal{P}_0 \), we relax the discrete variable \( l_k \) into continuous variable \( \tilde{l}_k \). The problem is relaxed as

\[
\mathcal{P}_1 : \quad \max_{\{d_k, \tilde{l}_k, f_k, \forall k \in K\}} Q \quad s.t. \\
C_1, C_2, C_3, \quad \tilde{l}_k^{min} \leq \tilde{l}_k \leq \tilde{l}_k^{max}, \forall k \in K.
\]

(4)

According to the observations in Sec. III, the objective function \( Q \) is non-decreasing with respect to \( d_k \) and \( l_k \). Therefore, we adopt the cyclic block coordinate gradient projection (CBGP) method to solve problem \( \mathcal{P}_1 \) [28]. According to the method, we solve problem \( \mathcal{P}_1 \) by fixing two of three variables and deriving the remaining one. We iterate the process until the value of each variable converges.

Denote \( \nabla y(x) \) as the partial derivative of function \( y \) corresponding to variable \( x \). Define \( P_{\Omega}(x) = \arg \min_{y \in \Omega} \|x - y\|^2 \).
as the Euclidean projection of $x$ on $\Omega$. The procedures of the solution can be summarized as follows:

- Given $\hat{l}_k$ and $f_k$, we update $d_k$ according to
  \[ d_{k}^{(j+1)} = P_{\Omega_d} \left( d_k^{(j)} + \alpha_k \nabla Q_k \left( d_k^{(j)} \right) \right), \forall k \in \mathcal{K}; \tag{5} \]
  where $\alpha_k > 0$ is a constant step size and $\Omega_d$ is the bounded domain constrained by $C_3$.
- Given $d_k$ and $f_k$, we update $\hat{l}_k$ according to
  \[ \hat{l}_k^{(j+1)} = P_{\Omega_l} \left( \hat{l}_k^{(j)} + \beta_k \nabla Q_k \left( \hat{l}_k^{(j)} \right) \right), \forall k \in \mathcal{K}; \tag{6} \]
  where $\beta_k > 0$ is a constant step size and $\Omega_l$ is the bounded domain constrained by $C_4$.
- Given $\hat{l}_k$ and $d_k$, the problem is simplified to
  \[ \max_{\{f_k, \forall k \in \mathcal{K}\}} Q \]
  \[ \text{s.t.} \quad C_1 : T_k \leq T_k^{\text{max}}, \forall k \in \mathcal{K}; \]
  \[ C_2 : \sum_{k \in \mathcal{K}} f_k \leq F^{\text{max}}; \tag{7} \]
  where constraints $C_3$ and $C_4$ are irrelevant to this problem.

We utilize the Lagrangian dual decomposition method to solve the above problem. The Lagrangian function is
\[
\mathbb{L}(f_k, \lambda, \mu) = Q + \mu \left( \sum_{k \in \mathcal{K}} f_k - F^{\text{max}} \right) + \sum_{k \in \mathcal{K}} \lambda_k \left( \frac{s(d_k)}{r_k} + \frac{p(l_k)}{f_k} + T - T_k^{\text{max}} \right).
\]

We then iteratively obtain $d_k$, $\hat{l}_k$, and $f_k$ until the algorithm converges. Since $\hat{l}_k$ is a relaxed mAP of the computation model, it may not match the mAP of any installed computation models. However, the GPU virtualization incurs the computation overhead and delay. Since the adaptive MAR optimization algorithm runs on a small time scale, e.g., every second, it is inefficient to dynamic generate vGPUs according to the optimization results. Therefore, instead of reconfiguring vGPUs, we assign clients’ requests to pre-configured vGPUs according to the optimization results.

Denote $g_i$ and $\mathcal{I}$ as the total computation resources of the $i$th vGPU and the set of vGPUs, respectively. Let $f_k$ be the computation resources allocated to the $k$th client. We formulate the client-vGPU mapping problem as
\[
\min_{\{a_{i,k}, \forall k \in \mathcal{K}, i \in \mathcal{I}\}} \sum_{i \in \mathcal{I}} \left( g_i - \sum_{k \in \mathcal{K}} a_{i,k} f_k \right) \text{s.t.} \]
\[ C_1 : g_i - \sum_{k \in \mathcal{K}} a_{i,k} f_k \geq 0, \forall i \in \mathcal{I}; \tag{13} \]
\[ C_2 : \sum_{i \in \mathcal{I}} a_{i,k} = 1, \forall k \in \mathcal{K}; \]
\[ C_3 : a_{i,k} = \{0, 1\}, \forall i \in \mathcal{I}, k \in \mathcal{K}; \]
where $a_{i,k}$ is an indicator function. If the $k$th client is assigned to the $i$th vGPU, $a_{i,k} = 1$; otherwise, $a_{i,k} = 0$. The client-
Algorithm 2: Greedy Client-vGPU Mapping

Input: \( f_k \) and \( g_i \), \( \forall k \in K, i \in I \).
Output: \( a_{i,k} \), \( \forall k \in K, i \in I \).
1 Sort \( f_k \) and \( g_i \) from the largest to the smallest;
2 \( a_{i,k} \leftarrow 0 \), \( \forall k \in K, i \in I \);
3 for \( k = 1 : |K| \) do
4     for \( i = 1 : |I| \) do
5         if \( f_k \leq g_i \) then
6             \( a_{i,k} = 1 \) and \( g_i = g_i - f_k \);
7                 break;
8     if \( \sum_{i \in I} a_{i,k} = 0 \) then
9         Set \( a_{j,k} = 1 \) where \( j = \arg \max_{i \in I} g_i \);
10        \( g_i = g_i - f_k \);
11 return \( a_{i,k} \).

The vGPU mapping problem is equivalent to a bin packing problem which is NP-complete. We solve this problem with a greedy approximation algorithm whose pseudo code is shown in Alg. 2. In this algorithm, we sort \( f_k \) and \( g_i \) based on their values from the largest to the smallest, respectively. Then, we sequentially assign a client to the vGPU which has sufficient computation resources to serve the client. If no vGPU can serve the client, the algorithm assigns the client to the vGPU with the most residual computation resources.

V. FRAME RATE ADAPTATION ON MOBILE CLIENT

The impact of the AR frame rate on the system performance is implicitly considered in the optimization problem \( \mathcal{P}_0 \). We assume that the MAR client continuously sends video frames to the edge server. Therefore, the frame rate is determined by the service latency, i.e., the frame rate of the \( k \)th client \( \omega_k = 1/T_k \). However, based on this frame rate, the computation resources allocated to the \( k \)th client are not fully utilized because the server has to wait for the frames from the client. To solve this problem, we design a frame rate adaptation mechanism on the MAR client to perform the traffic flow control in the system. This mechanism enables the MAR client to fully utilize the allocated computing resources on the edge server and helps them to increase the amount of virtual content obtained from the server without impairing the service latency.

As shown in Fig. 8, the frame rate adaptation determines how many AR video frames are sent to the edge server per second. With a higher video frame rate, the MAR client can send more frames to the edge server and thus obtain more virtual content. However, if the video frame rate is too high, sending and processing these frames may lead to the traffic congestion in wireless transmissions and the computation congestion on edge server. On the other hand, if the video frame rate is too low, the server waits for the frames from MAR clients which results in a under-utilization of computing resources. Therefore, the frame rate adaptation mechanism determines the AR frame rate based on the estimation of the computation workloads on the edge server. Since the computation workloads are varying, the AR frame rate also changes over time. In the system, we adapt the AR frame rate based on the computation latency. Hence, the frame rate of the \( k \)th client is calculated as

\[
\omega_k = \frac{1}{\gamma_k T_k^p}
\]

where \( T_k^p \) is the computation latency of the \( k \)th client, and \( \gamma_k \) is a control factor determined by the network conditions and computation workloads. A transceiving pipeline is designed to ensure that the sampled AR video frames are sent to the server continuously without waiting for the server’s response.

VI. PERFORMANCE EVALUATION

In this section, we provide an extensive evaluation of the DARE protocol under varying wireless channel conditions and computation workloads.

A. Protocol Implementation

Edge Server: A database is implemented to store clients’ information such as the minimum compression factors, wireless data rates and the latency requirements. This database will be updated based on the information received from the clients. The system periodically optimizes its performance and determines the AR configurations and computation resource allocations. Beside the periodical optimization, three events can trigger the optimization. They are 1) a newly connected client, 2) an optimization request from a client, and 3) a disconnected client. We establish long-standing sockets between clients and the edge server to exchange the control information such as the optimization request, network status report, and AR configurations.

Based on the client-vGPU mapping, clients are assigned to a vGPU using different socket ports. The computation resources on a vGPU are shared among the clients associated with the vGPU. We use multithreading to serve multiple clients simultaneously in the system. When a client connects to the system, a thread is created to serve the client. Using this thread, the edge server receives the client’s video frames and pushes them into the computation queue. The tasks in the computation queue are scheduled according to the adaptive MAR optimization algorithm.
MAR Client: The QoA monitor on the MAR client is implemented to monitor the wireless channel conditions and the service latency. The QoA monitor will trigger the optimization request to the edge server if the wireless channel condition changes or the service latency exceeds $T_{\text{max}}$. An optimization request contains the information about the wireless data rate, $T_{\text{max}}$, and $d_{\text{min}}$. We evaluate the wireless channel condition and service latency every second to avoid generating excessive optimization requests based on the service latency of an individual frame.

The wireless channel conditions of clients are important for optimizing the edge-based MAR system. To effectively estimate wireless data rates, we use the Linux wireless-tool iwconfig to fetch received signal strength indicator (RSSI) from the driver of the wireless network adapter. Then, we map the RSSI to the wireless data rate.

B. Evaluation Setup

We deploy the edge server on a workstation with an Intel Xeon E5-2630v4 2.2GHz CPU, Nvidia Quadro M4000 and P2000 GPUs, and 16GB RAM. The features of the GPUs installed in our edge server are detailed in Table I. The MAR clients are deployed on laptops. We emulate the AR video using VOC 2007 test dataset [9] that contains 4952 images. Since we do not have video dataset for evaluating the mAP of different visual object detection algorithms, we construct a video with 4952 images in VOC 2007 test dataset to evaluate the proposed protocol. The images in the dataset are resized to 500x500 pixels. We use the resized images to generate AR video streams. The clients are connected to the edge server via a LinkSys WRT1900AC wireless router. In the experiments, the static latency $T_c$ of the edge-based MAR system is 0.04 s. Based on measurements, we derive the mapping between RSSIs and wireless data rates as shown in Table II.

Table I. GPU details

<table>
<thead>
<tr>
<th>Function</th>
<th>NVIDIA Quadro M4000</th>
<th>NVIDIA Quadro P2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA cores</td>
<td>1664</td>
<td>1024</td>
</tr>
<tr>
<td>Memory size</td>
<td>8 GB</td>
<td>5 GB</td>
</tr>
<tr>
<td>Boost Clock</td>
<td>800 MHz</td>
<td>1470MHz</td>
</tr>
<tr>
<td>Power</td>
<td>120 W</td>
<td>73 W</td>
</tr>
<tr>
<td>Peak FP32 Performance</td>
<td>2.66 TFLOPS</td>
<td>3.0 TFLOPS</td>
</tr>
</tbody>
</table>

Table II. RSSI vs. Data Rate

<table>
<thead>
<tr>
<th>RSSI (dBm)</th>
<th>-30</th>
<th>-40</th>
<th>-50</th>
<th>-60</th>
<th>-70</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Rate  (Mbps)</td>
<td>25</td>
<td>20</td>
<td>15</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

On the edge server, we deployed five computation models based on the YOLO framework. These models are YOLO 288x288, YOLO 352x352, YOLO 416x416, YOLO 480x480, and YOLO 544x544. Here, YOLO 288x288 means that the YOLO object detection algorithm tuned at the image resolution of 288x288. The mAPs of these models are listed in Table III where we use 288x to represent YOLO 288x288. On the MAR client, the range of the image compression factors are from 0.4 to 1.0.

We compare the performance of the DARE protocol with two baseline protocols:
- **Max QoA**: this system uses the high-fidelity video data and high-complexity computation model to maximize mAP of the object detection for MAR clients.
- **Min Latency**: this system uses the low-resolution video frames and low-complexity computation model to minimize the service latency experienced by MAR clients.

C. Evaluation Results

Wireless channel conditions: To evaluate the performance of the DARE protocol under dynamic wireless channel conditions, we maintain a static workload on the edge server and carry the MAR client randomly walking in the laboratory. During the random walk, the MAR client experiences different wireless channel conditions. Fig. 9 compares the performance of the DARE protocol with two baseline protocols. As shown in the figure, the Max QoA protocol has the longest service latency but provide the highest QoA. The Min Latency protocol minimizes the service latency at the cost of a significantly decreased QoA. For most of the frames, the DARE protocol maintains the MAR client’s minimum latency requirement which is 0.3 s in the experiment. At the same time, the DARE protocol maximizes the QoA of the MAR client considering various wireless channel conditions. The QoA achieved by the DARE protocol is only a slightly smaller than that achieved by the Max QoA protocol.
Fig. 10 shows that the DARE protocol tracks the wireless channel conditions and adapts the video frame size and computation model accordingly. Fig. 10 (a) shows the wireless received signal strength indication (RSSI) and the corresponding service latency of the MAR client in the duration of 4952 AR video frames. When RSSI drops, there are a few latency spikes in the system. This is because the DARE protocol performs the optimization every second. If RSSI changes between two consecutive optimizations, the DARE protocol does not respond to the changes immediately. The optimization interval can be configured in the system. A small optimization interval improves the performance of the system in tracking wireless channel variations but also increases the optimization workloads on the server.

As shown in the Fig. 10 (b), the video frame size is adapted more frequently than the computation model. This observation indicates that the DARE protocol prefers to adapting the video frame size first when RSSI changes. For most of the frames, the DARE protocol uses the high-complexity computation model to ensure a high QoA and only degrades the computation when the wireless channel condition is very poor.

Server workloads: Fig. 11 shows the performance of the DARE protocol with different number of MAR clients. In this experiment, the clients are stable so that their wireless channel conditions are almost static. When there are more clients in the system, the computation latency increases because all the clients share the limited computation resources on the server. For both the Max QoA and Min Latency protocols, their service latency almost linearly increases with the number of clients. The DARE protocol maintains a low service latency that satisfy the latency requirements (0.3 s) of MAR clients. The cost of maintaining the low latency is dropping QoA of the object detection of MAR clients. However, the DARE protocol is able to mitigate the tradeoff between QoA and the service latency through optimizations. For example, when the number of user is 3, the DARE protocol trades about 0.06 QoA for about 31% latency reductions as compared with the Max QoA protocol.

Service latency requirements: In this experiment, we vary $T_{max}$ to evaluate the performance of the DARE protocol under different latency requirements. Fig. 12 shows that for most of the time, the DARE protocol can satisfy the latency requirements of the mobile client. Under a stringent latency requirement, e.g., 0.1 s, the DARE protocol serves the client with a low QoA at about 0.68.

Number of computation models: The available computation modes on the server impact the performance of the adaptation. To study such an impact, we evaluate the performance of the DARE protocol under three settings of the computation models. In the first setting, we install one computation model, YOLO 416x416, on the server. In the second second setting, we install three computation models: YOLO 325x352, 416x416 and 480x480. In the third setting, we install five computation models: YOLO 288x288, 325x352, 416x416, 480x480 and 544x544.

The experiment results are shown in Fig. 13. When there is only one computation model on the edge server, the DARE protocol can only adjust the client’s video frame size. Since the service latency is smaller than the latency requirement which is $T_{max} = 0.2$ in the experiment, the DARE protocol adopts the largest video frame size to maximize QoA. As a result, under this scenario, the DARE protocol has the similar performance as the Max QoA protocol. The Min Latency protocol adopts the minimum image data size to minimize the server latency. Hence, its service latency is smaller, but its QoA is also lower than the DARE protocol and the Max QoA protocol.

When there are three computation models in the system, the

![Figure 10: The adaptation to wireless channel conditions.](image)

![Figure 11: The protocol performance with different number of users.](image)

![Figure 12: The performance of DARE with different $T_{max}$.](image)
service latency of the Max QoA protocol is still less than the latency requirement of the client (0.2 s). Hence, the DARE protocol adopts the maximum video frame size and the high-complexity computation model to maximize its QoA. When there are five computation models in the system, the service latency of the Max QoA protocol is larger than the latency requirement of the client (0.2 s). Under this scenario, the DARE protocol maximizes QoA with the constraint of the client’s latency requirement. Therefore, the service latency of the DARE protocol nearly equals to 0.2 s, and the QoA is slight less than that of the Max QoA protocol. For the Min Latency protocol, it always selects the low-complexity computation model with smallest video frame size. Therefore, the service latency as well as QoA of the Min Latency protocol decreases when a computation model with a lower complexity is available in the system.

**Video frame rate and pipelining**: With the DARE protocol, the MAR client dynamically changes the AR frame rate according to workloads on the edge server. Fig. 14 shows the impact of the video frame rate on the server workload. In the experiment, the service latency is about 0.3 s. Without the dynamic video frame rate mechanism (baseline), the AR video frames are sent after receiving the virtual content from the server. Therefore, the client can only send about 3 AR video frames to the server per second which is determined by the service latency. With the adaptive frame rate mechanism, the MAR client can send about 16 AR video frames per second on average to the server. In Fig. 14(b), with the adaptive frame rate mechanism on the MAR client, the GPU utilization on the server is almost 100%. This indicates this mechanism is able to exploit the allocated computation resources on the server to maximize the frame rate of the MAR client.

**VII. Conclusion**

In this paper, we have designed the DARE protocol to provide high quality MAR service with edge computing. The unique feature of the DARE protocol is that it can dynamically adapt the AR configurations and the computation resource allocations on the edge server according to the wireless channel conditions and computation workloads. We have studied the tradeoff between the quality of augmentation and service latency in the edge-based MAR system and built analytical models to characterize such a tradeoff. We have developed optimization mechanisms to guide the AR configuration and server computation resource allocation. We have implemented the DARE protocol and validated that the protocol ensures the service latency of MAR clients and maximize the quality of augmentation under varying network conditions and computation workloads.

**ACKNOWLEDGEMENT**

We appreciate insightful comments and suggestions from the shepherd and the anonymous reviewers. This work is partially supported by the U.S. National Science Foundation under Grant No. 1731675, No. 1810174 and the UNC-Charlotte Faculty Research Grant.

**REFERENCES**


