EdgeSync: Efficient Edge-Assisted Video Analytics via Network Contention-Aware Scheduling

Qiang Xu*
School of Electrical and Computer Engineering
Purdue University, West Lafayette, USA
Email: xu1201@purdue.edu

Ravi K. Rajendran, Murugan Sankaradas, and Srimat T. Chakradhar
NEC Laboratories America, Inc.
Princeton, USA
Email: {rarajendran, murugs, chak}@nec-labs.com

***Abstract*—With the advancement of 5G, edge-assisted video analytics has become increasingly popular, driven by the technology's ability to support low-latency, high-bandwidth applications. However, in scenarios where multiple clients competing for network resources, network contention poses a significant challenge. In this paper, we propose a novel scheduling algorithm that intelligently batches and aligns the offloading of multiple video analytics clients to optimize both network and edge server resource utilization while meeting the Service Level Objective (SLO). Experiment with a cellular network testbed shows that our approach successfully processes 93% or more of inference requests from 7 different clients to the edge server while meeting the SLOs, whereas other approaches achieve a lower success rate, ranging from 65% to 85% under the same condition.**

***Index Terms*—Edge Computing, Video Analytics, Network Contention, Adaptive Scheduling, 5G**

I. Introduction

The advancement of 5G, including local/private 5G, provides low latency, high bandwidth, security, and scalability for large-scale video analytics applications that require real-time processing of large amount of data on computationally heavy tasks such as object detection and tracking. This high-speed network capability enables edge computing, where tasks are offloaded from end devices to nearby edge servers, reducing latency and lowering the computational and energy demands on client devices. Edge computing is especially valuable for applications in environments like smart factories, healthcare, and augmented reality, where continuous, high-resolution video analytics must be processed in near real-time [1], [2]. For such applications, Service Level Objectives (SLOs) are defined to specify acceptable latency thresholds, ensuring timely processing and high-quality user experiences.

To maximize throughput, adaptive batching dynamically groups inference requests based on real-time network and server conditions, optimizing latency and resource utilization. Unlike fixed-size batching, adaptive batching adjusts the batch size to accommodate stochastic requests and meet the SLOs.

While prior works [3]–[5] have extensively explored the benefits of batching to optimize GPU inference performance, they often overlook the critical challenge of network contention. In scenarios where multiple clients compete for limited network resources, network contention can significantly degrade application performance. This is particularly true for edge-assisted video analytics applications where end-to-end latency is crucial [6]–[9]. Network contention leads to increased latency and jitter, making it difficult to meet stringent Service Level Objectives (SLOs). Addressing this challenge requires a holistic approach that considers both batching and network conditions to ensure efficient resource utilization and satisfying the SLO.

To this end, we design EdgeSync, an offloading scheduling system to address the crucial issue of network contention that impacts concurrent real-time video analytics applications. EdgeSync dynamically aligns frame capture and offloading timings, minimizing request drop rates and efficiently utilizing GPU resources. The system is structured around three key components: an offloading alignment profiler that assesses optimal timings for various user configurations, an alignment planner that dynamically adapts capture schedules as the number of users fluctuates, and a frame timing controller that precisely adjusts capture timing on the devices. This approach ensures EdgeSync can meet stringent service-level objectives while adapting to changes in network load and device availability.

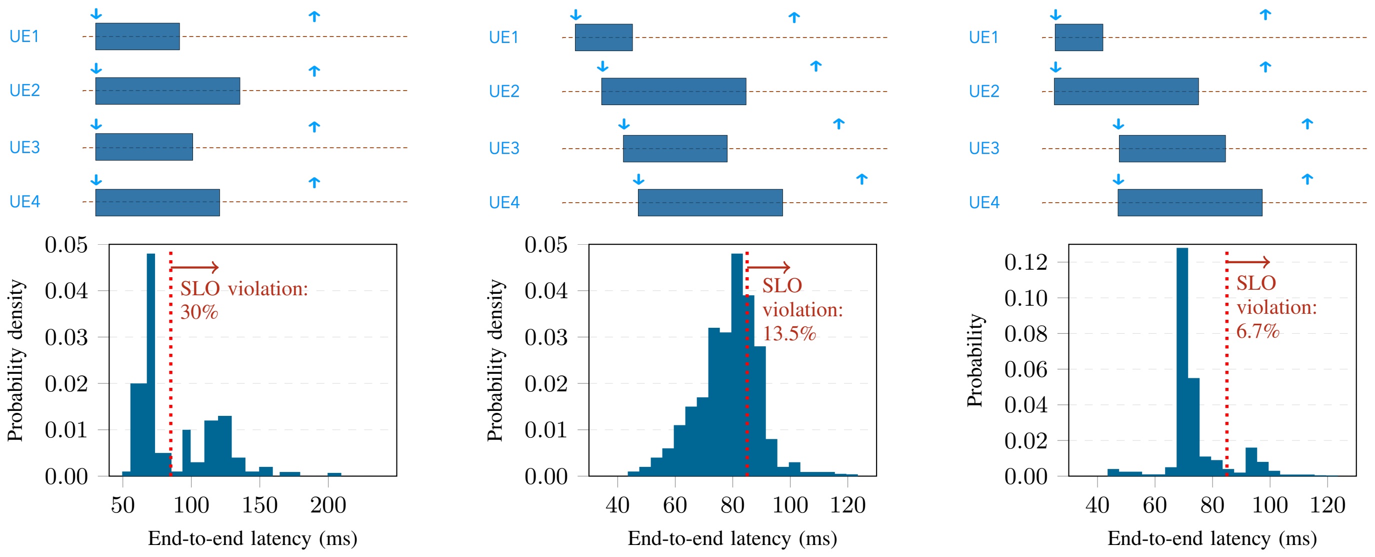
Our experimental evaluation using a private cellular testbed demonstrates significant improvements in processing success rates and latency management under network contention. EdgeSync processed 93% or more of inference requests from 7 different user equipment (UE) [1] devices while maintaining the SLO. In comparison, alternative approaches that lacked EdgeSync's intelligent scheduling and alignment mechanisms saw considerably lower success rates, ranging from 65% to 85%, under the same network conditions. The results validate EdgeSync's capability to optimize both edge server resource utilization and network efficiency.

In summary, the paper makes the following contributions:

- We identified the critical impact of network contention among multiple clients running edge-assisted video analytics applications.
- We implemented EdgeSync, a novel system that integrates adaptive batching and network-aware frame timing control to optimize resource utilization while satisfying SLO.
- We experimentally validated EdgeSync on a private cellular testbed, demonstrating its ability to maintain a high processing success rate and meet the SLO.

*Work done as an intern at NEC Laboratories America, Inc.

[1]Throughout the paper, we use the terms "client" and "UE" interchangeably.



(a) Fetch all together (b) Evenly spaced fetch (c) Fetch group of 2 UEs

Fig. 1: Dynamic adjustment of frame capture times across User Equipments (UEs)

II. Related Work

Prior works on edge-assisted video analytics [3]–[5], [7], [10], [11] mainly focus on adaptive batching to improve the GPU throughput for latency-constrained tasks by adjusting batch sizes and resource allocation on the server side. The idea of adaptive batching is initially proposed by Clipper [4], while Nexus [3] scales the concept to a cluster of servers running multiple video analytics tasks. MCDNN [11] further exploits shared model prefixes to avoid redundant computation. While these works obey the latency SLO into account during scheduling, only the server inference latency is taken into account. However, the network latency, which plays a significant role in the end-to-end latency, is neglected by these works. Critically, these methods do not account for network contention, a major factor in multi-client scenarios where bandwidth competition can significantly degrade latency and SLO adherence.

III. Problem Statement

Prior works have explored in-depth optimizing the edge server's throughput of processing video analytics task requests with adaptive batched inference. However, real-time video analytics requires end-to-end latency, which includes both the execution latency on the edge server and the transmission latency incurred by the network, to meet the tasks' latency requirements. Yet the network transmission latency, which contributes to a major part of the end-to-end latency, is often ignored by prior works.

To address this gap, we ask the question: How to optimize the performance of edge-assisted video analytics clients in the presence of network constraints? Answering this question requires us to consider the following factors:

- **The network latency:** The network latency is the time it takes for a data packet to travel from the client to the edge server. This latency can vary depending on a number of factors, such as the distance between the client and the edge server, the congestion of the network, and the network type.
- **The impact of network contention:** When there are multiple video analytics clients requesting services from the same edge server, the network can become congested. This can increase the network latency for all clients, which can degrade the performance of the video analytics application.

IV. Motivation

A. *Impact of Network*

To study the impact of the network on edge-assisted video analytics applications, we conducted two experiments.

In the first experiment, we generated object detection requests at a rate of 140 requests per second. We let the GPU server process the requests with adaptive batched inference [3]. The detailed setup, e.g., DNN model and GPU, is described in Section VII-A. In this experiment, there was no consideration of the network. There were also no request drops, meaning that all requests were successfully processed.

In the second experiment, we connected 7 Raspberry Pi devices to initiate requests at 20 requests per second each. The total request rate was 140 requests per second, as in the first experiment. The devices were connected to the server through a Celona private cellular testbed. In this experiment, the request drop rate was 16.8%. Due to the additional network latency, the adaptive batching algorithm chooses smaller batch sizes to accommodate for the shorter latency budget left for inference, and the smaller batch size instead leads to less efficient GPU utilization and thus lower throughput. Because of this, some requests are dropped to accommodate for the lowered server capacity (a request is dropped when it is backlogged for a long time and can no longer meet its SLO). Above experiments confirm network layer has a major impact on video analytics application performance.

B. Impact of Network Contention

Batched inference, which is essential to the high utilization of the GPU server, requires multiple requests (from different clients) to arrive before the inference starts, which means that requests that arrive early need to wait for later requests. The waiting time, which also contributes to the end-to-end latency, in turn affects the adaptive batching decisions due to the shorter latency budget left, as discussed before.

To minimize the queuing time, one approach is to control all clients from the batch and make them all send their requests at the same time, as shown in Fig. 1a. However, such an offloading scheme leads to as high as 30% latency SLO violations with an SLO threshold of 85 ms. Further investigation shows that when offloading the requests together, the contention of network bandwidth leads to longer frame transfer latency. In contrast, to minimize network contention, one could space the frame transfer out (Fig. 1b), which, however, leads to longer waiting time and thus high SLO violation rate (13.5%) as well.

We instead found that mixing the two approaches together, i.e., dividing UEs into groups and offloading requests from the same group together while spacing different groups out (Fig. 1c), can strike a balance between waiting latency and transfer latency, and achieve a lower SLO violation rate (6.7%).

V. Challenges

While the approach discussed above effectively reduces the SLO violations, there are various challenges in making this approach to work in real systems.

a) Dependency on system setup: The optimal offloading alignment for real-time video analytics applications depends on various factors, such as: The complexity of the DNN model and the associated inference times; the latency SLO requirements, which specify the maximum acceptable latency for the application; and the total number of user equipment present in the system. The offline profiling phase can take into account of certain static factors, such as the complexity of the DNN models and the number of UEs. However, it is not possible to predict the access point's scheduling policy or the varying number of clients in the online serving phase.

b) Varying number of clients: Edge orchestration is used to optimize the use of resources at the edge. This includes things like scheduling inference tasks, caching DNN models, and managing communication between the edge and the cloud. Device orchestration is used to control the sensing timing of UEs. This is important because the network transfer time is not deterministic, so we need to try to control when the frames are captured to minimize latency. The online serving phase is responsible for dynamically adapting to system changes, such as the arrival of new UEs or the departure of existing UEs. This is done by continuously monitoring the system and adjusting the offloading alignment accordingly.

VI. EdgeSync Architecture

Addressing the above challenges requires a technique that can automatically identify the optimal inference batching size

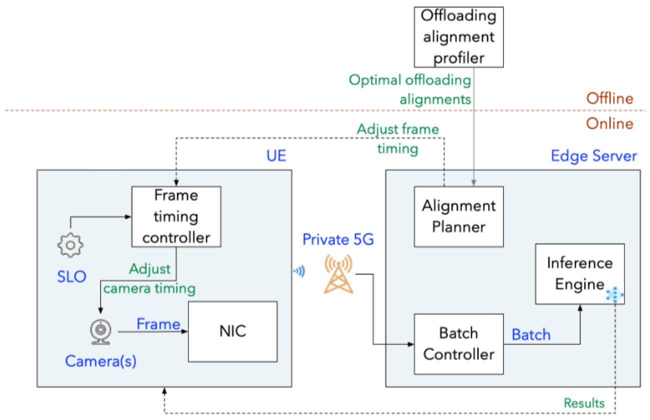


Fig. 2: Proposed system architecture

and sensor fetch timing, and dynamically adjust between them. To this end, we design EdgeSync, which consists of three major components: (a) *offloading alignment profiler*: offline profiling determines the optimal frame timings under different numbers of UEs (b) *alignment planner*: dynamically calculates how to adapt between plans as the number of UEs changes (c) *frame timing controller*: executes the adjustment controlling the camera timing. The overall system architecture is shown in Fig. 2.

A. Offloading Alignment Profiler

Profiling plays a crucial role in the optimization of complex systems, especially when undergoing configuration changes such as transitioning to a new DNN model or altering network configurations. This process enables the comprehensive analysis of system performance under varying conditions, ensuring that the system maintains its efficiency and responsiveness. For instance, in the context of augmented reality applications, where low latency is quintessential, profiling becomes indispensable when modifying system parameters like DNN models or network configurations. It provides insights into how these changes impact task execution times, resource utilization, and overall system behavior. By continually profiling, we can fine-tune the system to ensure seamless user experiences even as configurations evolve, safeguarding against potential performance bottlenecks that might arise from such alterations.

Here , profiling is done to capture the relationship between the number of UEs and group size. These factors are crucial due to their direct influence on frame capture timings and subsequently, the user experience. The number of UEs impacts the timing at which frames are captured, a crucial element in maintaining low latency. Furthermore, different group sizes necessitate distinct frame timing plans, as frames from UEs within the same group are aligned, while frames from different groups are evenly distributed. The heart of the profiling process lies in assessing the request drop rate, a metric that gauges the efficiency of frame capture timing plans. By profiling this parameter across varying UE counts and group sizes, developers gain an intricate understanding of how different system configurations impact request drop rates, allowing them

Algorithm 1 Frame timing profiler

```
function FRAME_TIMING_PROFILER(ue_count,group_size)
    for ue_size = 1 to ue_count do
        for grp_size = 1 to group_size do
            total_grp_count = ⌈ue_size / grp_size⌉
            ▷ assign UEs to target groups
            a) requests from UEs of same group are aligned
            b) requests from UEs of different groups are evenly spaced out
            feed requests to batched inference engine
            measure request drop rate & inference time
        end for
        CACHE best grp_size for current ue_size
    end for
end function
```

Algorithm 2 Frame timing controller

```
function FRAME_TIMING_CONTROLLER
    Receive frame timing adjustment request from the frame timing planner.
    Determine the direction (advance or delay) and magnitude of the adjustment ($k$ ms).
    ▷ Apply the incremental adjustments to the camera's frame capture timing for $k$ consecutive frames.
    for $i$ = 1 to $k$ do
        if adjusting forward then
            Delay next frame's capture time by 1ms
        else
            Advance next frame's capture time by 1ms
        end if
    end for
end function
```

to tailor frame timing plans to achieve optimal performance for each UE count. This iterative approach ensures that the best frame timing plans are preserved, even as the number of UEs and group sizes change, guaranteeing consistently high-quality and low-latency video analytics QoE. The frame timing profiler steps are provided in Algorithm 1.

B. Alignment Planner

The frame capture timing planner addresses the challenge of seamlessly adjusting user equipment from one frame timing plan to another, especially when UEs are added or removed. The central objective of this planner is to minimize the total amount of UE timing change during such transitions. The goal is to orchestrate the frame capture times in a way that minimizes disruptions to e.g., interactive applications like Augmented Reality.

The methodology consists of two key steps. First, a cost matrix C, is constructed, where each cell denotes the required frame capture time change when migrating from one position to another. This matrix encapsulates the temporal adjustments needed for a smooth transition. The problem then transforms into a linear assignment problem, which is solved using the Hungarian algorithm. The algorithm seeks the minimum sum of costs by selecting a single element from each row, while adhering to the constraint that chosen elements must belong to distinct columns. Consequently, the cells chosen by the algorithm dictate the optimal adjustments for the UEs, ensuring that the transition minimizes the overall timing changes.

The alignment planner's main objective is to achieve smooth transitions for UEs in frame capture timing plans, with the constraint that total UE timing change should be minimized. The planner ensures that the selections made by the algorithm result in the most effective and efficient adjustments, enabling the UEs to be integrated or disengaged from the frame timing plan with minimal impact on the overall system. This proactive approach to UE timing optimization enhances the quality and consistency of the user experience, particularly in dynamic scenarios involving changes in the number of UEs.

The objective function can be formulated as follows.

$$\min \sum_{\substack{i \in \mathbf{U} \\ j \in \mathbf{G}}} C_{ij} X_{ij} \tag{1}$$

subject to

$$\sum_{i \in \mathbf{U}} X_{ij} = 1 \quad \forall\, j \in \mathbf{G} \tag{2}$$

$$\sum_{j \in \mathbf{G}} X_{ij} = 1 \quad \forall\, i \in \mathbf{U} \tag{3}$$

where C is the cost matrix, $X_{ij} \in \{0,1\}\ \forall\ (i \in \mathbf{U},\ j \in \mathbf{G})$ is a binary matrix and $\mathbf{U} = \{u_1, u_2, \ldots, u_m\}$ is set of UEs and $\mathbf{G}$ is the group sizes for the DNN model $\mathbf{D_x}$. The steps for frame capture timing alignment planning are as follows:

1) Initialize the cost matrix based on the required frame capture time changes for UEs' transitions.
2) Apply the Hungarian algorithm to the cost matrix to identify the optimal UE adjustments that minimize total timing changes.
3) For each UE group, batch the requests for frame capture adjustments.
4) Optimize the batched requests to streamline resource usage.
5) Fine-tune the captured adjustments for each UE, ensuring they adhere to constraints such as minimum and maximum timing change limits.
6) Execute the finalized frame capture timing adjustments, seamlessly transitioning the UEs while minimizing disruptions.

C. Frame Timing Controller

Finally, the frame timing controller implements the frame capture time adjustments determined by the frame timing planner in the client. Upon receiving the timing adjustments from the planner, the frame timing controller takes charge of coordinating the camera's timing to effectuate the recommended changes. This ensures that the seamless transitions and optimal UE adjustments suggested by the planner are accurately executed in the device's operation. Notably, the controller operates based on fine-grained control APIs that

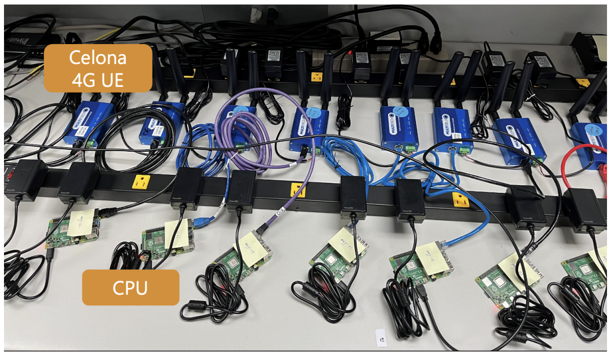


Fig. 3: Experimental setup with Raspberry Pi devices connected to Celona 4G LTE UE

are commonly available on edge devices like Android and HoloLens [12], [13], enabling precise management of frame capture timing.

One key optimization strategy employed by the frame timing controller is mini-step adjustment. In this approach, when the server instructs a UE to advance or delay its requests by a certain amount, for instance k ms, the frame timing controller increments or decrements the capture time of the next k frames, each by just 1 ms. This granular adjustment strategy contributes to smoother transitions between different time slots. Given that most frame timing changes are typically within the range of <10 ms, this technique ensures that adjustments can be accommodated within a short span of time, approximately 10 frames or 167 ms. The controller is explained in Algorithm 2.

This procedure shows how the frame timing controller manages the frame capture timing adjustments incrementally, aligning with the guidance from the frame timing planner. This approach ensures that transitions between different timing slots are executed with precision.

VII. Performance Evaluation

A. Experimental Setup

Our experiment setup focuses on establishing an end-to-end system for real-time augmented reality applications. Both client and server functionalities are implemented in Python, showcasing a comprehensive approach to tackling the challenges of real-time processing and synchronization.

On the server side, the system leverages Nvidia's TensorRT for efficient Deep Neural Network (DNN) inference. The Hungarian algorithm, crucial for alignment tasks, is implemented using SciPy's implementation. This pairing of technologies on the server side ensures optimal DNN performance and accurate task assignment.

The client side of the experiment involves Raspberry Pi devices equipped with HD cameras, simulating video analytics devices. Frames are captured utilizing the Picamera2 library, with the added capability to read frames from stored data for reproducible evaluations. Prior to transmission, frames are compressed into JPEG format and transmitted to the server. The server, in turn, decompresses these frames before initiating DNN inference. Moreover, the experiment incorporates the

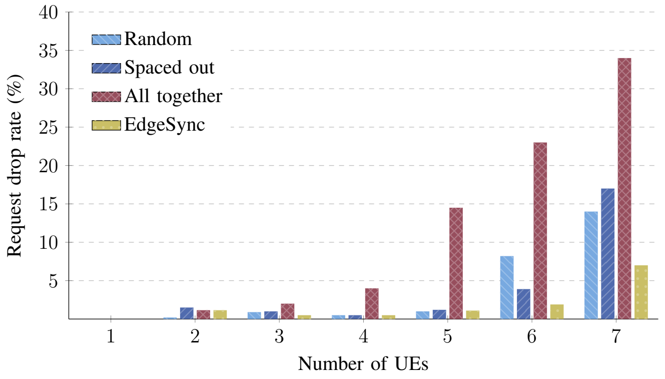


Fig. 4: Comparison of request drop rates with varying number of UEs under different approaches

usage of the chrony protocol to synchronize time between the server and clients. This synchronization enables precise time measurements, allowing the server to gauge the time elapsed between frame availability on the client and frame reception on the server. This chrony-based synchronization mechanism ensures that temporal evaluations are accurate, with time synchronization errors typically kept below 1 ms under the considered network conditions.

The edge component of this setup operates as a single server responsible for executing both DNN inference and alignment planning. Featuring an x86 CPU and an Nvidia 2070 GPU, the edge server is empowered to handle the computational demands of real-time video analytics applications. The DNN model of choice is DETR [14] with a ResNet 101 backbone, highlighting the system's capability to accommodate complex models. All Raspberry Pi-4B devices, serving as video analytics serving development kits, share a uniform architecture. In total, eight of these development kits are all interconnected through a Celona 4G LTE network infrastructure as shown in Fig. 3. This network configuration reflects a simplified direct connection between clients and the edge server, minimizing routing complexities and enhancing data transmission efficiency.

Baselines. We compare proposed design, EdgeSync, with three other methods (a) randomly placed: not controlling the frame capture time (b) spaced out: evenly space out the frame capture times among the UEs and (c) all together: align frame capture times of all UEs to the same time. Edge server resources are not over-subscribed in any of the scenarios.

B. Results

a) Effectiveness of EdgeSync: We evaluated the effectiveness of our system under different numbers of UEs linked to the edge server and gauging request drop rates across different setups. Fig. 4 shows the results of EdgeSync, achieving the lowest request drop rate consistently as the number of UEs increases as opposed to other non-alignment techniques. Our system can process the requests successfully above 93%. This substantial improvement underscores the potency of EdgeSync in ensuring consistent and uninterrupted

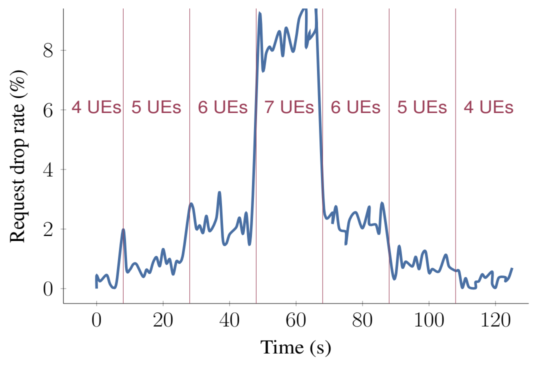


Fig. 5: Request drop rate of EdgeSync as the number of UEs vary over time

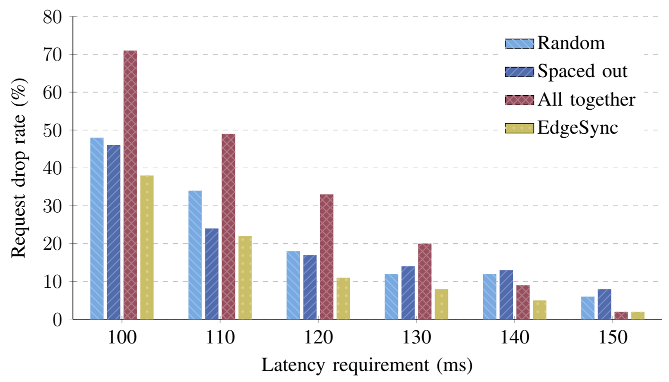


Fig. 6: Sensitivity analysis with varying latency requirements

service delivery, thereby affirming its viability in dynamic and demanding operational scenarios.

b) Adaptiveness to fluctuating load: We studied how EdgeSync adapts to varying load. We progressively increase the number of user equipment connected to the edge server at intervals of 20 seconds. Conversely, we initiate a reduction in the number of UEs after the count reaches seven. This adaptive approach ensures seamless transitions between different UE counts and their associated frame timing plans. Notably, UEs autonomously align themselves with the altered plans as the UE count changes. The system diligently tracks the request drop rate over time to gauge performance under varying conditions. Importantly, these adaptive processes are executed without causing disruptions to ongoing UE requests. Fig. 5 shows that our system demonstrates adaptability to fluctuating loads through dynamic adjustments.

c) Sensitivity analysis: We also conducted analysis on how the SLO is honored by our system. Each request sent from UE contains latency SLO requirements in addition to frame data and some other metadata. To do the evaluation, the number of UEs is preset to 7 and fixed throughout the experiment. The SLO is changed from stringent (100 ms) to relaxed (150 ms) and the average request drop rate is measured under various methods. EdgeSync is able to achieve the lowest request drop rate under different latency requirements as shown in the Fig. 6.

VIII. Conclusion

In this paper, we proposed a novel batching algorithm and intelligent scheduling algorithm for offloading real-time video analytics tasks to edge servers. Our algorithms take into account both Service Level Objective (SLOs) and available network bandwidth to minimize the latency and energy consumption of video analytics applications. We also evaluate our proposed algorithms using real-world cellular testbed. Our results show that our algorithms can significantly improve the performance of multi-client video analytics applications in terms of latency and resource utilization. In the future, we plan to extend our work to consider other factors that can affect the performance of video analytics applications, such as the quality of the network connection.

References

[1] Y. Ma, C. Lu, B. Sinopoli, and S. Zeng, "Exploring edge computing for multitier industrial control," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[2] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019.

[3] H. Shen, L. Chen, Y. Jin, L. Zhao, B. Kong, M. Philipose, A. Krishnamurthy, and R. Sundaram, "Nexus: A gpu cluster engine for accelerating dnn-based video analysis," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019.

[4] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A Low-Latency online prediction serving system," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017.

[5] F. Romero, M. Zhao, N. J. Yadwadkar, and C. Kozyrakis, "Llama: A heterogeneous & serverless framework for auto-tuning video analytics pipelines," in *Proceedings of the ACM Symposium on Cloud Computing*, 2021.

[6] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. Netravali, "Reducto: On-camera filtering for resource-efficient real-time video analytics," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2020.

[7] V. Nigade, R. Winder, H. Bal, and L. Wang, "Better never than late: Timely edge video analytics over the air," in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, 2021.

[8] D. Xu, A. Zhou, G. Wang, H. Zhang, X. Li, J. Pei, and H. Ma, "Tutti: coupling 5g ran and mobile edge computing for latency-critical video analytics," in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, 2022.

[9] A. Padmanabhan, N. Agarwal, A. Iyer, G. Ananthanarayanan, Y. Shu, N. Karianakis, G. H. Xu, and R. Netravali, "Gemel: Model merging for Memory-Efficient, Real-Time video analytics at the edge," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023.

[10] C. Olston, F. Li, J. Harmsen, J. Soyke, K. Gorovoy, L. Lao, N. Fiedel, S. Ramesh, and V. Rajashekhar, "Tensorflow-serving: Flexible, high-performance ml serving," in *Workshop on ML Systems at NIPS 2017*, 2017.

[11] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, 2016.

[12] (2017) Mediacapture.getpreviewframeasync method. [Online]. Available: https://learn.microsoft.com/en-us/uwp/api/windows.media.capture.mediacapture.getpreviewframeasync?view=winrt-22621

[13] (2017) Cameracapturesession. [Online]. Available: https://developer.android.com/reference/android/hardware/camera2/CameraCaptureSession

[14] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," 2020.