Energy-aware Mobile Edge Computing for Low-Latency Visual Data Processing

Huy Trinh, Dmitrii Chemodanov, Shizeng Yao, Qing Lei, Bo Zhang, Fan Gao, Prasad Calyam, Kannappan Palaniappan
University of Missouri-Columbia, USA
Email: {hntzq4, dycbt4, syyh4, qlzm3, bz7v2, fgyf8}@mail.missouri.edu; calyamp@missouri.edu, pal@missouri.edu

***Abstract*—New opportunities exist for applications such as disaster incident response that can benefit from the convergence of Internet of Things (IoT) and cloud computing technologies. Particularly, new paradigms such as Mobile Edge Computing (MEC) are becoming feasible to handle the data deluge occurring in the network edge to gain insights that assist in real-time decision-making. In this paper, we study the potential of MEC to address application issues related to energy management on constrained IoT devices with limited power sources, while also providing low-latency processing of visual data being generated at high resolutions. Using a facial recognition application that is important in disaster incident response scenarios, we analyze the tradeoffs in computing policies that offload visual data processing (i.e., to an edge cloud or a core cloud) at low-to-high workloads, and their impact on energy consumption under different visual data consumption requirements (i.e., users with thick clients or thin clients). From our empirical results obtained from experiments with our facial recognition application on a realistic edge and core cloud testbed, we show how MEC can provide flexibility to users who desire energy conservation over low-latency or vice versa in the visual data processing.**

***Keywords*—IoT-based applications, cloud computing, energy awareness, visual data processing, mobile edge cloud**

I. Introduction

The Internet of Things (IoT) is becoming increasingly relevant for innovations in smart city applications such as manufacturing and public safety. Mobile devices, wearable smart devices and sensors are being connected with diverse network connectivity options (e.g., austere infrastructure, Gigabit network speeds at the network edge), and applications can benefit from the insights in the data from these IoT devices. Especially for applications such as disaster incident response or law enforcement, visual data (e.g., high-resolution images, video clips) from IoT devices needs to be processed in real-time. Relevant insights from the data can help incident commanders to quickly analyze scenes and deploy resources (e.g., paramedics, ambulances, medical supplies) [1]. Through convergence with cloud computing technologies, IoT-based application data can be handled at large scale from multiple network edge sites with on-demand computation capabilities.

However, it is not always reasonable to assume that fully functional computing/networking infrastructure, and unlimited power sources exist to handle the visual data processing needs. In natural disaster situations involving earthquakes, hurricanes, or man-made disasters involving terrorism, edge infrastructure may be lost and computation for disaster incident response decision-making might require relying on constrained mobile devices in terms of computing, networking or power resources. One important IoT-based application we can envisage that is useful involves facial recognition technology, which provides fast and accurate identification when high-resolution image data, and high-performance computing/networking exist to match against a large/distributed database of images. The identification can help find 'lost persons' or identify 'bad actors' [2] in disaster scenes, if achieved in real-time and through processing of a high volume of images at the network edge on a limited power budget.

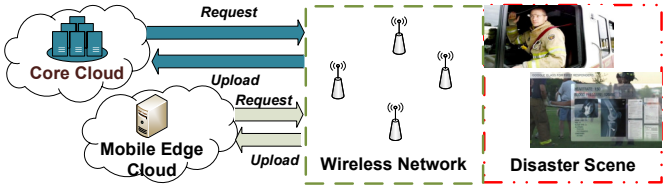


Fig. 1: Illustration of disaster scene related visual data processing by use of wireless network, mobile edge and core cloud resources to upload images and request processed images using sophisticated computer vision algorithms.

Figure 1 shows an illustration of how new paradigms of Mobile Edge Computing (MEC) [3], [4] are emerging that allow for upload of raw images and download requests of processed images in the exemplar facial recognition application context. MEC architectures allow for distributed computing in Radio Access Networks (RANs) by having cellular operators to cooperate application developers and content providers. Using MEC, we can augment critical infrastructure by having the cloud computing resources more distributed and accessible close to the wireless network-edge. For instance, it allows for a base station infrastructure or 'cloudlets' to handle computation requests from mobile devices that are in the geographic vicinity [5], [6]. This provides options to offload computation tasks from IoT devices to address application issues related to energy management on constrained IoT devices with limited power sources, while also providing low-latency processing of visual data using sophisticated computer vision algorithms. However, there is a need for better understanding on the MEC paradigm potential in terms of its benefits or limitations when edge clouds are used with a core cloud that may have: (a) undesirably long round-trip times, (b) intermittent connectivity, or (c) excessive congestion, as in the case of austere or adverse network edge environments.

In this paper, we aim to study the potential of the MEC paradigm by using the context of a facial recognition ap-

This material is based upon work supported by the National Science Foundation under Award Number: CNS-1647182. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

plication in a disaster incident response scenario. Our goal is to adopt MEC within the facial recognition application framework and analyze the tradeoffs in computing policies that offload visual data processing (i.e., to an edge or a core cloud) at low-to-high workloads, and their impact on energy consumption under different visual data consumption requirements.

As part of our paper contributions, we particularly consider visual data consumption for users with *thin client* or *thick client* configurations; thin client configuration at a user assumes all of the processed images are stored and viewed at a remote cloud resource, whereas thick client configuration assumes processed images are downloaded and further post-processed at the mobile user device level. When available, we assume the core cloud has the option to provide multiple compute instances which can help in *parallel* processing of visual data workloads, versus having limited edge cloud resources that process the workloads in a *sequential* manner. Further, we consider cases where compression is used in the image transfers, which could save bandwidth consumption in austere networks, but increases the energy consumption that could have a negative impact on the power-constrained IoT device or edge cloud side with limited power sources.

To provide a flexible option for IoT-based applications to decide whether to offload the visual data processing to an edge cloud or a core cloud for the above user requirement cases, we present a novel 'decision-making algorithm'. Our algorithm handles cases where a hard real-time processing need exists or a varying scale of visual data processing workload needs to be handled at the network-edge, while meeting user requirements that are energy conscious or demand fast processing.

We evaluate our energy-aware and low-latency MEC framework featuring the facial recognition application and our decision-making algorithm with experiments in a realistic edge and core cloud testbed. For the edge cloud, we use a campus server, and we use the GENI Cloud resource [7] for the core cloud. We leverage the Android-based PowerTutor utility [8] to profile and estimate energy consumption (Metric: Joules) of our facial recognition application that is based on OpenCV [9] within the testbed. Our experiment results show how MEC can provide flexibility to users who desire energy conservation over low-latency (Metric: Processing Time) or vice versa in visual IoT-based application data processing. We compare cases where using thin client or thick client configurations are more effective at low-to-high visual data processing workloads, and how offloading policies could affect the energy efficiency or low latency user requirements.

The remainder of paper is organized as follows: Section II reviews related work. In Section III, we present our facial recognition application and a MEC framework for studying computation offloading policies to balance tradeoffs in energy efficiency and low-latency processing of low-to-high scale workloads from IoT devices. Realistic GENI Cloud testbed experiments and results discussion are described in Section IV. Section V concludes the paper.

II. Related Works

Computation Offloading Decision-Making. Existing literature on computation offloading can be classified under two categories of work. First set of works such as [10], [11] consider the concept of "program partition", which involves offloading parts of a given processing task onto edge servers, and other parts of the task run on user devices. Specifically, they propose offline heuristic algorithms to support a large-scale mobile application and thereby reduce the completion time for all application users. A second set of works, such as [12], [13] consider a "migration" strategy that offloads the entire application onto an edge server. Specifically, the authors in [12] create a device classification for prioritizing computation that is based on the channel and base station resource allocation status. In [13], the authors use a Markov decision process to dynamically offload computation within services. If offloading is not a viable option, authors in works such as [14], [15] propose "load shedding", a prevalent data-stream management technique. Load shredding involves automatically either dropping or adapting the quality of packets on the edge device. Our work differs from existing works due to the energy-awareness and low-latency user requirements handling we address that flexibly allows visual data processing to occur either at the edge cloud or in the core cloud depending on the tradeoffs involved.

Visual Data Consumption. To display visual data from a remote system, it is common to use either thin client or thick client solutions. A thin client [16] can typically run on local computer hardware (e.g., keyboard, mouse, display) that is able to remotely connect to a remote desktop that is either cloud-hosted or on a remote server. The computation burden in this case will reside on the server side, and screen scrapes are sent to the client. A thick client, on the other hand, can be assumed to be a fully functional computer or device that possess computing resources that are significant for post-processing visual data based on user drill-down or zoom in/out. According to [17], a stateless thick client might still require periodic connection and computation assistance from the cloud or a remote server. Regardless, user satisfaction in terms of image rendering quality and interaction depends on the session latency that depends on the network bandwidth and computational resources at the client/server sides. The authors in [18] found from real-world measurements that even with good bandwidth of 100 Mbps, the latency in thin clients still falls in range of 33-100 ms across different cities. Moreover, they recommend the use of "cloudlet" or "Mobile Edge Computing" architectures as a suitable solution to lower end-to-end latency. Our work builds upon this recommendation in our visual data processing workflow that is part of the MEC architecture based facial recognition application.

III. Energy-aware and Low-latency MEC Framework

In this section, we first describe the facial recognition technology and our application framework implementation that is important in disaster incident response when used by incident commanders and first responders. Following this, we detail our computation offloading decision-making algorithm that can handle scalable workloads and energy constraints of IoT devices that use our facial recognition application.

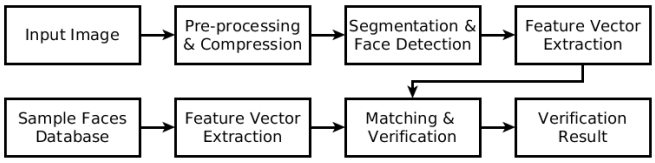


Fig. 2: Overview of steps in facial recognition for target identification.

A. Application Background and Implementation

Facial recognition technology when used in an application on a mobile device can help in identifying or verifying a person's identity whose digital image is collected from a local camera/video source. The facial recognition process we use in our work has several steps as shown in Figure 2 that involve the digital image at the client side and a larger image sample dataset at the server side.

To initially detect a human face for a given database of images within a small amount of time (i.e., with low latency), we perform a pre-processing step during which we compress all the input images. After compression, the face region in each given image is detected and segmented using eigenfaces techniques described in [19]. With the segmented face information, we apply feature extraction using a Histograms-of-Oriented-Gradients (HoG) [20] feature detector and save all of the extracted features into a target feature vector. Once we have the target feature vector information, we perform matching between target feature vector and multiple feature vectors of sample candidates in the sample image database such that each candidate in the database will have a matching score. In our implementation, this step could consume long processing times, especially when there are a large number of candidate images in the database. After calculating all the individual matching scores, we choose the candidate with the highest matching score in the final verification result.

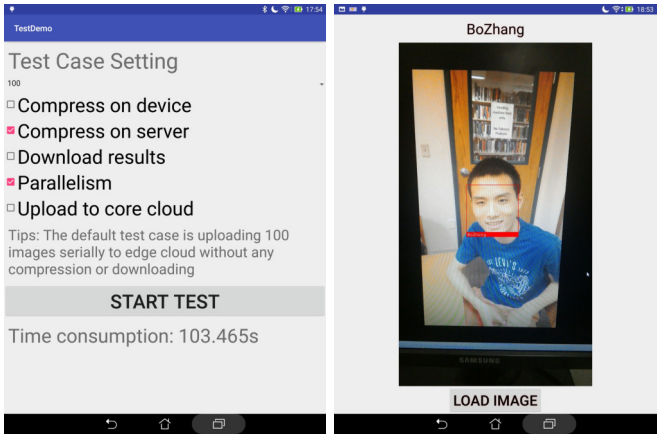


Fig. 3: Application GUI - Left Image: test options for user to select, Right Image: received result from the server side.

Figure 3 shows the graphical user interface of our application implementation that is developed for an Android device using the Java programming language. The facial recognition process described above has been implemented using OpenCV [9] for image management and using Python scripts that utilize Dlib [21], an open source library. To use this interface and obtain, for instance, the name of the matched image, a user can choose different computation and image transfer polices as shown in the right half of Figure 3 such as: compression on device or server, thin or thick client, serial or parallel processing. The resulting image of the target identification along with processing time consumption can be obtained from the server side as shown in the left half of Figure 3 for single or multiple image uploads from one or more IoT devices simultaneously.

B. Computation Offloading Decision-Making Algorithm

The thin client or thick client application simply sends the data from the mobile device to a cloud server to achieve better results in low-latency processing and the related energy consumption. However, the decision between offloading computation to the edge or core cloud depends on the user requirement and workload scale. Authors in [5] show that the edge cloud improves response time from 200ms to 80ms and energy consumption reduced by 30%-40%. However, the core cloud is helpful because of unlimited resources and parallel instances to speedup processing. Therefore, we propose an algorithm to classify the scenario with the user's choice to help choose the best visual data processing decision.

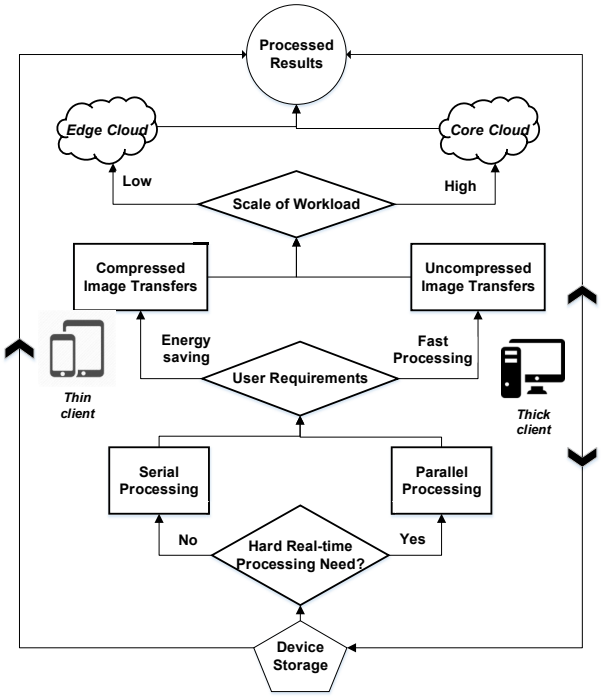


Fig. 4: Illustration showing the flexible policies-based computation offloading decision-making.

The application logic of the image/photo processing is displayed as Figure 4. After the photo is captured, there are multiple decisions that will make a difference on the energy consumption and processing latency. For instance, the transformation of the photos can be performed in parallel or in

a sequential/serial manner, depending on real-time processing needs of the users. In addition, the photos can be compressed before being uploaded to the cloud platform. Obviously, the photo size will be smaller and thus it takes less bandwidth to upload but at the expense of additional energy and time consumption. Without comparison and analysis, it is challenging to decide whether the overall effect is positive or negative. The same problems arise when the results are sent back to the device based on user requirement in the thick client case. Our experiments seek to evaluate the tradeoffs in these various conditions under low-to-high workload scales. The real-time requirement of the application is another factor, i.e., if the face recognition results are required instantly for post-processing on the client side, the workflow has to be optimized. However, if the face recognition results are not required instantly, the results can be simply shown on the server instead of transferring the results back to the client side. In this way, redundant steps can be eliminated and a better performance can be achieved based on the user requirements as well as the client/server capabilities.

Workload allocation to the edge cloud or core cloud considering energy awareness introduces additional challenges for various scenarios. For example, due to a case where energy conservation and fast computation time are desired, processing has to be completed using a cloud platform. However with the remote processing, the additional energy consumption to transfer images also affects the processing latency. Computation offloading onto the edge cloud in this case could save energy and image transfer time, however the edge cloud might have limited resources to handle large workload scales or facilitate parallel processing. The energy and latency metrics thus should be given different priority (or weight) for different workloads so that a reasonable strategy can be selected in the end-to-end steps of the visual data processing.

Algorithm 1 shows our energy and latency aware steps in computation offload colorbluedecision-making. The *main()* function gets executed first to check whether the user needs to receive the final results from the server as in the thick client case; or whether thin client assumptions are relevant on the user side. Once a decision on either the thin or thick client is made, two operations occur subsequently. Firstly, *create_Thread()* ensures that the mobile device is creating multiple threads to start UDP sessions if we have multiple server instances provisioned in the core cloud as represented by variable *number_of_servers*. Based on the need, the value of *number_of_servers* can be configured for parallel or serial transfer to server. Consequently, serial transfers frequently consume less energy, but result in longer processing times. Secondly, function *Offload_decision()* chooses among 'with' or 'without' image compression options for sending the data to the edge or core cloud depending on the scale of workload and the real-time processing user requirement. If the workload is large which means either the number of images or their resolution is high, the mobile devices will send data directly to the core cloud. Note that image compression will help if the workload is low and the user requires energy conservation. Moreover, to avoid the overloading the edge cloud, the mobile device could periodically monitor edge/core cloud resources and check for availability before transferring data.

Algorithm 1: Offload decision-making

```
Data: Device info: RAM, processor, memory_capacity
Data: Load info: resolution, size_of_load
Data: User requirements: Download, Battery_Saving
Result: The efficient way to save energy and achieve
        low-latency processing
function create_Thread ()
    /* Create multiple threads on the mobile device to send
     load balanced data to different servers for processing */
     send ←
     device_thread_create(processor, {size_of_load,
     number_of_servers});
end
function Offload_decision (EdgeServer)
    /*Decide when and where to offload computation and
     transfer data */
     if User_Requirement() then
        compress();
     end
     if Workload.isLow() then
        sendToEdge();
     else
        sendToCore();
     end
     transmit();
end
function main ()
    /* Decide best client configuration */
     if Download_result then
        Thick_client();
     else
        Thin_client();
     end
     create_Thread();
     Offload_decision();
end
```

IV. Performance Evaluation

In this section, we evaluate our energy-aware and low-latency MEC framework featuring the facial recognition application and our decision-making algorithm with experiments in a realistic edge and core cloud testbed.

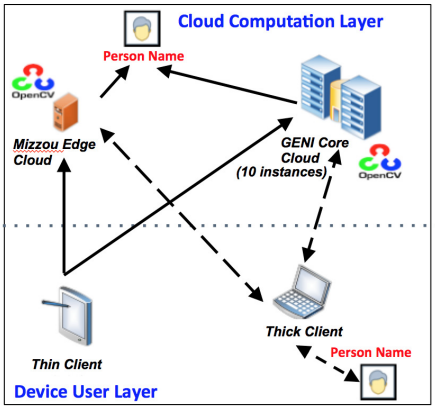


Fig. 5: Testbed edge cloud is a server on Mizzou campus and our core cloud includes 10 GENI server instances at New York University (NYU) campus.

Experimental Settings Figure 5 shows our experimental testbed setup where we use a local U. of Missouri server resource [7] for an edge cloud, and we use 10 GENI server instances at New York University (NYU) campus for the core cloud. Our edge cloud server has 70GB of RAM, 12 cores with a bandwidth of approximately 90 Mbps. Each of the core cloud servers have 1 core and 1 GB of RAM, and we connect to them at a bandwidth of approximately 900 Mbps. We use ASUS Zenpad tablet with 2 GB RAM, 1.33 GHz Atom Z3735 processor and 8 hours of battery life (when under common use) as our mobile device that runs the facial recognition application described in Section III.

Comparison methods and metrics. We compare cases where *thin* client or *thick* client configurations are used. Particularly, our thin client configuration at a user assumes all of the processed images are stored and viewed at a remote cloud resource, whereas a thick client configuration assumes processed images are downloaded and further post-processed at the mobile device level. We start our experiments by offloading application threads on the mobile device to the cloud computation layer for remote processing as shown in Figure 5. We vary the processing workload from 100 to 1000 images of 2048 x 1536 pixels size that are transferred to the remote server side using the UDP protocol. We also use different MEC policies including parallel (Par) processing versus serial or sequential (Seq) and data compression (C) versus no compression (NC) policy. We use the Android-based PowerTutor utility [8] to profile and estimate *energy consumption* of our facial recognition application (Metric: Joules) within the testbed setup. Our end-to-end *processing time* includes the time needed for an image export/import to edge or core cloud and its remote processing (Metric: Processing Time in Seconds).

A. Policy-based Optimization

We start our MEC framework evaluation by discussing its optimal policy sets (defined by a combination of decision parameters in Figure 4) that cover diverse user's demands and input data scale. Based on energy consumption results in Figures $6a$ and $6e$, we observe the following policies needed for the *maximum operational time of the mobile device*: for both *thin* and *thick* client configurations, we need to use *parallel* processing over *non-compressed* data. This observation is due to the fact that data compression significantly utilizes CPU resources of the mobile device, and the sequential data offloading further introduces additional energy consumption for data export/import. Note however in this case, offloading either to the edge cloud or core cloud has no impact on the energy consumption within the mobile device. However, processing time results shown in Figures $6b$ and $6f$ indicate how our optimal MEC policies for the *minimum end-to-end processing time* have changed. Particularly, for both *thin* and *thick* client configurations, we now need to use *parallel* processing over *compressed* data. Moreover, when the workload scale is low (e.g., $\leq$ 500 images), we can further speed up our remote processing by offloading to the edge cloud versus offloading to the core cloud. This difference is due to the higher latency of transferring data to the cloud, which degrades as the workload scale increases; in this case, the edge cloud needs more time to process all the data than the core cloud. Note also how in both cases, a *sequential* offloading policy is worth simultaneously for both the energy consumption and the processing time benefits. However, this policy is needed for live image data (e.g., video streams), where new frames are sequentially captured. Moreover, for both thin and thick client configurations, improving interactivity require more energy consumption in all cases.

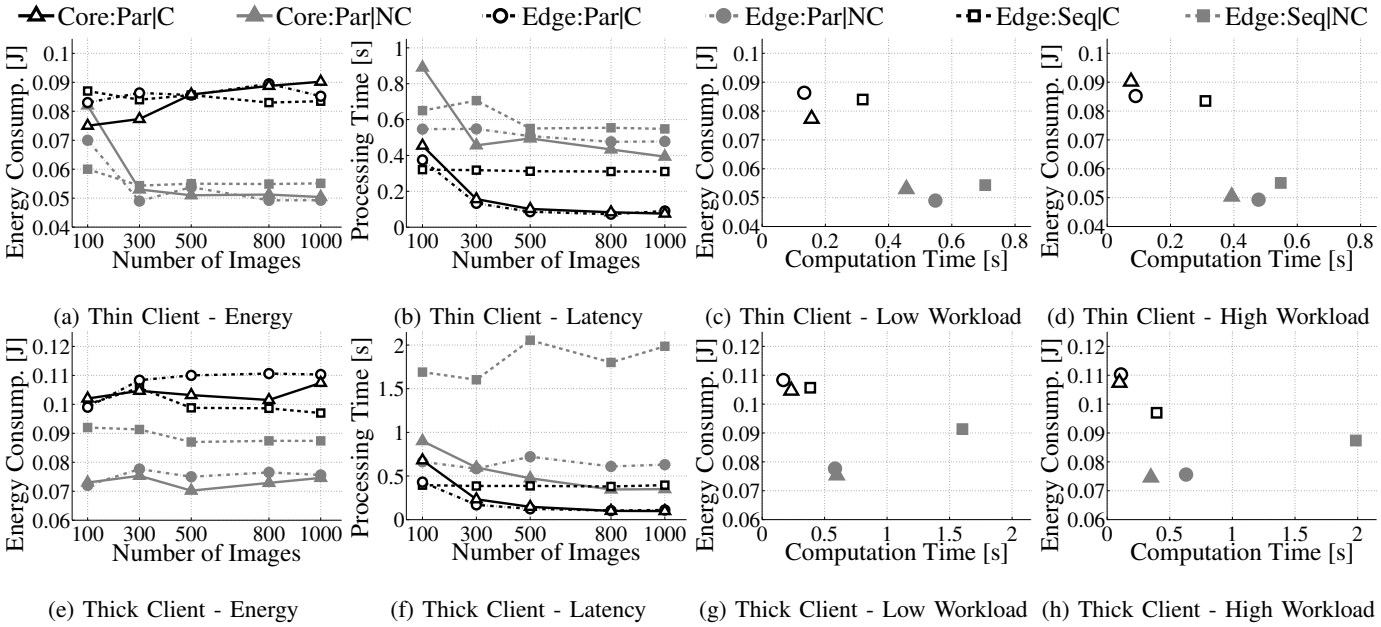
B. Engineering Trade-offs and Pareto Optimality

In practice, users can also benefit from considering an acceptable performance for the reasonable energy consumption instead of only focusing on a single factor as discussed previously. Below, we show how different policy selections can be a part of the Pareto optimal MEC framework strategy for different application energy consumption and processing time trade-offs. Specifically, when observing Figures $6c$ and $6g$ of a low workload scale (i.e., when processing $\approx$300 images), we can see how *parallel* processing of both compressed and non-compressed data at the edge or core cloud are part of the Pareto optimality for both *thin* and *thick* client configurations. More concretely, using *thin* client as an example, *parallel* processing of both compressed and non-compressed data at the core cloud could be among the top two optimal solutions. Overall, both of them could achieve relatively low energy consumption and short end-to-end computation time, but each of them has special advantages. Processing with compressed data consumes 65% less computation time compared to processing of non-compressed data, which requires 46% more energy consumption. Meanwhile, processing with non-compressed data may cost 31.5% less energy consumption, but it takes 191.4% more computation time for end-to-end process. In certain situations, users could choose the optimal solution based on their specific processing demands. To sum up, all of these policy combinations do not result in application performance cases that simultaneously degrade in both the energy consumption and the end-to-end processing time aspects.

The same however does not hold for a high data workload (i.e., when processing $\approx$1000 images). Particularly, observing Figures $6d$ and $6h$, we noticed how *parallel* processing of both compressed and non-compressed data at the edge cloud is not part of the Pareto optimal solution set when a *thick* client configuration is used. In this scenario, using *parallel* processing of non-compressed data at core cloud is the optimal solution since it has the lowest energy consumption and third shortest end-to-end computation time (only consumes 83% more computation time compared with the shortest solution but saves 49% energy consumption). The reason for this result is because of the fact that the edge processing time dominates higher data export/import latencies to the cloud. Thus, computation offloading to the edge cloud under a high data workload for *thick* clients is always suboptimal to the core cloud offloading for our facial recognition application context.

V. Conclusion

In this paper, we studied how the mobile edge computing (MEC) paradigm can provide flexibility to users who desire energy conservation over low-latency or vice versa in visual IoT-based application data processing. Our work was based on



(a) Thin Client - Energy (b) Thin Client - Latency (c) Thin Client - Low Workload (d) Thin Client - High Workload

(e) Thick Client - Energy (f) Thick Client - Latency (g) Thick Client - Low Workload (h) Thick Client - High Workload

Fig. 6: Average energy consumption (a,e) and processing time (b,f) per image with their trade-offs under a low workload of 300 images (c,g) and under a high workload of 1000 images (d,h) on the IoT device using thin (see *top*) and thick (see *bottom*) clients when offloading to the Core or Edge clouds with parallel (Par) or sequential (Seq) processing policies as well as with data compression (C) or no data compression (NC) policies.

the rationale that computing should happen in the proximity of data sources, and cloud services especially moved closer to the network edge can present opportunities to meet user requirements in terms of energy consumption and fast processing times. Using a facial recognition application that we developed for use on mobile devices, we were able to demonstrate cases where thin client or thick client configurations are more effective at low-to-high visual data processing workloads, and how offloading policies could affect the energy efficiency or low latency user requirements. Particularly, we found from the results that the edge cloud offloading policy for thick clients is always sub-optimal in comparison to the core cloud offloading under high workloads. However, it was not the case for thin clients under similar conditions.

For future work, we plan to investigate advanced algorithms to make computation offloading decisions considering multiple edge cloud resources to satisfy user requirements in visual data processing involving other constraints such as user mobility.

REFERENCES

[1] J. Gillis, P. Calyam, O. Apperson and S. Ahmad, "Panacea's Cloud: Augmented reality for mass casualty disaster incident triage and coordination", *IEEE Annual Consumer Communications and Networking Conference (CCNC)*, pp. 264-265, 2016.

[2] J. Klontz, A. Jain, "A case study on unconstrained facial recognition using the Boston marathon bombings suspects, *IEEE Computer*, 2013.

[3] A. Ahmed, E. Ahmed, "A Survey on Mobile Edge Computing", *International Conference on Intelligent Systems and Control*, pp. 1-8, 2016.

[4] Y. Jararweh, A. Doulat, O. AlQudah, E. Ahmed, M. Al-Ayyoub, E. Benkhelifa, "The Future of Mobile Cloud Computing: Integrating Cloudlets and Mobile Edge Computing", *International Conference on Telecommunications (ICT)*, pp. 1-5, 2016.

[5] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, M. Satyanarayanan, "Towards Wearable Cognitive Assistance", *Proc. of International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pp. 68-81, 2014.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge Computing: Vision and Challenges", *IEEE Internet of Things Journal*, Vol. 3, No. 5, pp. 637-646, 2016.

[7] NSF GENI Infrastructure - http://www.geni.net

[8] L. Zhang, B. Tiwana, R. Dick, Z. Qian, Z. Mao, Z. Wang, L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones", *Proc. of IEEE/ACM/IFIP Intl. Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pp. 105-114, 2010.

[9] G. Bradski, A. Kaehler, "Learning OpenCV: Computer vision with the OpenCV library", *O'Reilly Media*, 2008.

[10] G. Orsini, D. Bade, W.Lamersdorf, "Computing at the Mobile Edge: Designing Elastic Android Applications for Computation Offloading", *IFIP Wireless and Mobile Networking Conference (WMNC)*, pp. 112-119, 2015.

[11] L. Yang, J. Cao, H. Cheng, Y. Ji, "Multi-user computation partitioning for latency sensitive mobile cloud applications", *IEEE Transactions on Computers*, Vol. 64, No. 8, pp. 2253-2266, 2015.

[12] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, Y. Zhang, "Energy-Efficient Offloading for Mobile Edge Computing in 5G Heterogeneous Networks", *IEEE Access*, Vol. 4, pp. 5896-5907, 2016.

[13] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, K. Leung, "Dynamic Service Migration in Mobile Edge-Clouds", *IFIP Networking Conference*, pp. 1-9, 2015.

[14] D. Andersson, P. Elmersson, A. Juntti, Z. Gajic, D. Karlsson, L. Fabiano, "Intelligent load shedding to counteract power system instability", *Proc. of IEEE/PES Transmission and Distribution Conference and Exposition: Latin America*, pp. 570-574, 2004.

[15] N. Perumal, A. C. Amran, "Automatic load shedding in power system", *Proc. of National Power Engineering Conference*, pp. 211-216, 2003.

[16] A. Alesheikh, H. Helali, H. Behroz, "Web GIS: Technologies and its Applications", *Symposium on Geospatial Theory, Processing and Applications*, Vol. 15, 2002.

[17] N. Tolia, D. Andersen, M. Satyanarayanan, "Quantifying Interactive User Experience on Thin Clients", *Computer*, Vol. 39, No. 3, pp. 46-52, 2006.

[18] M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing", *IEEE Pervasive Computing*, Vol. 8, No. 4, pp. 14-23, 2009.

[19] M. Turk, A. Pentland, "Face recognition using eigenfaces", *Proc. of IEEE Computer Vision and Pattern Recognition*, pp. 586-591, 1991.

[20] N. Dalal, B. Triggs, "Histograms of oriented gradients for human detection", *Proc. of IEEE Computer Vision and Pattern Recognition*, 2005.

[21] D. King, "Dlib-ml: A Machine Learning Toolkit", *Journal of Machine Learning Research*, Vol. 10, pp. 1755-1758, 2009.