**Automobile Traffic Detection on the Edge: A Performance Comparison**

Group 8 : *Beomjun Aaron Bae, Ritwik Nandakumar, Tootiya Giyahchi*

## 1. Introduction

Edge computing is a paradigm where the data and information computation is performed at the local edge node itself. With the increase in the computing powers of mobile units as well as the rise in popularity of IoT, a strategy to take advantage of these edge resources has been getting a lot of attention. No matter how load-resilient a server farm can become, there are bottlenecks that occur in a classical network of server-client systems. Instead, if the network architecture was modified so that computation can be offloaded to non-centralized servers, the bottleneck would be relieved and the quality of service would increase. The vision to offload the computation to the edge of the network has already grabbed the research and the industrial communities, and many projects have been proposed if not already implemented. Taleb et al proposes a smart city application of Mobile Edge Computing (MEC) where tourists will see a significant decrease in latency in accessing tourist videos by storing the videos on the edge [3]. Since the videos will only be accessed on devices near the tourist attraction, a local edge network is perfect for the application. Another example of unique, successful projects in edge computing is Hyrax. Hyrax is attempting to create an edge-only network that can be especially helpful in rescue emergency scenarios, extending the network reachability via peer-to-peer like scheme [4].

The advantages in adopting an edge computing architecture continues to be realized in different research efforts, but the difficulty in managing these edge networks still remains to be cumbersome. As Varghese et al mentions, challenges like security, efficient discovery of edge nodes, partitioning tasks, and offloading tasks are all deterrents to the ability to widely adopt edge computing networks [6]. There are countlessly many configurations and environments to manage given the sheer variability in available devices and the endlessly unique set of network requirements based on the problem at hand.

To manage these problems, many different platforms have been developed to streamline the development of edge computing networks. Open Networking Foundation developed an IoT management platform called CORD, and University of Wisconsin - Madison has developed their own, called ParaDrop. Besides these, there are many other variants like EdgeX, Edgegent, Firework, and Link Edge IoT. However, out of all these services, two of them are growing to be an industry standard: Amazon's AWS Greengrass IoT and Microsoft Azure IoT.

In our project, we have benchmarked the two platforms, AWS Greengrass and Microsoft Azure IoT, placing the edge device in a detection environment to classify and label images, followed by sending the data to the Cloud. We developed an edge to cloud architecture that classifies whether a supposed screenshot of a traffic footage contains an automobile or not. Then the classified results will be sent to the cloud to be stored in a centralized repository. With this architecture, we imagine that the system could be

implemented as a traffic-load sensor with minimal cost and low latency. Furthermore, to compare and contrast the differences between AWS Greengrass and MS Azure IoT, we measured some baseline statistics like end-to-end latency, resource utilization, and bandwidth usage. Note that the edge devices were selected to be a group of Raspberry Pis, because they are often used as a standard for low resource devices.

## 2. Related Work

While surveying the literature, we came across a large number of papers that aimed to check and perform comparative edge computing evaluations. [3], [4] and [5], which gives us edge compute motivations, a middleware architecture and edge compute scenario envisionings respectively, are all relevant to our edge computing implementation but too generalized to allow for a significant direct impact that aids our project.

Among the sources surveyed, papers that we found most pertinent to our project were [1] and [2]. The publication release [1], entitled "Towards a Methodology for Benchmarking Edge Processing Frameworks" offers a preliminary benchmark methodology for any given edge processing platform. This aids us in creating relevant evaluation criteria, benchmark objectives, and an edge workflow for our project. The authors in [1] provide an open source benchmark tool, called 'Edgebench', to compute and compare performance characteristics for different edge processing platforms. The list of tested frameworks includes Microsoft Azure Edge IoT and Amazon AWS Greengrass IoT. This work is also complemented by the authors own list of evaluation metrics.

In our project, we have utilized parts of the Edgebench test suite/tool to evaluate our own setup keeping in mind our project scenario needs. We have also tweaked the underlying code for metrics display to match our evaluation and performance requirements, in addition to using our own custom hardware and dataset for object classification. The Edgebench system tool is available for any interested third party to use, through a GitHub public repo. In our project, we have utilized this framework to test the edge processing /edge computation aspect of the system and omitted the testing of the cloud-only performance aspects of the suite due to time constraints. This, however, could be addressed by including it in future project iterations.

## 3. Goals & Methodology

### a. Goals

Our project scenario envisions the edge device present in a traffic scenario where it will be used as a sensory device to monitor the number of vehicles on the road. Immediately, the setting calls for not only a limited set of computing resources but also a wide-spread scalability if we were to adopt an edge device on every intersection of a city, for example. Seeing these requirements, we found Raspberry Pi as the computing edge node would be suitable, because it has low energy requirements along with powerful enough computing power to identify automobiles from a feed of images.

We pose this vehicle detection problem as an image classification problem. We hope to deploy a machine learning model to the edge nodes so that they will be able to identify whether a vehicle is detected on the image feed, and send that information to a central server. The model we believe that we chose was Apache MXNET. This model is able to detect and generate many different class labels, but adapted with a simple filtering mechanism, it will be able to detect automobiles. We will then feed the label and class information characteristics to the cloud which then performs an analysis of the received info to take the next step. We want to identify the overall latency overhead for Microsoft Azure IoT system and compare it to AWS Greengrass system in this image classification application. Thus, we also measure a few performance metrics such as response time, propagation time, and overall latency (for each platform).

To do so, we extended an existing benchmarking platform, EdgeBench, created by Anirban et al in 2018 [1]. EdgeBench benchmarks AWS GreenGrass and Amazon Azure IoT in two different network schemes. The first is an edge network where the computation is getting done locally at the end devices (Raspberry Pi 3B) and then the result is sent to the cloud, as shown in Figure 1 and 2. The second is an architecture where the computation is done on the cloud, but this is not a part of our project.

**b. Devices**

As we mentioned earlier, we used the Raspberry Pi 3B model as end devices. All devices are connected to the internet via a wireless router. To have them communicate via AWS Greengrass all devices had AWS IoT SDK installed and will run Greengrass core software so they can be managed and modified through the AWS console website. For Azure, we have an Iot EdgeAgent and IoT Hub installed on the RPi that communicates to the cloud, where the collected messages can be accessed through Azure's web interface. Greengrass authorizes and secures all the communication through the MQTT protocol. For MS Azure IoT, Docker compatible containers were installed for all edge modules so they can be managed and modified using Azure IoT Edge API. To keep our setup comparable to EdgeBench [1], we make use of their GitHub repo code base.

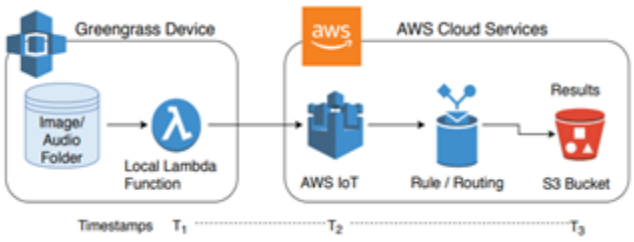


**Fig. 1 GreenGrass Architecture**

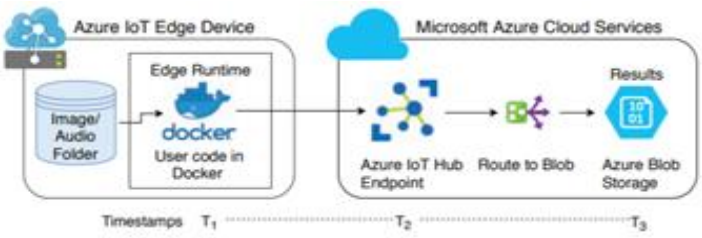


**Fig. 2 Azure IoT Edge Architecture**

### c. Application and Input Data

The image classification application is the same that was deployed in the Edgebench project with a few adjustments to fit our new dataset. In the original project, Edgebench used a raw image set collected from the ImageNet project. They consisted of variegated sizes of images that were crawled from the internet. However, to keep the scope of the project pinpointed, we needed a dataset that included images of automobiles. Hence, we use the CIFAR-10 dataset. CIFAR-10 dataset includes ten different categories of images, of which automobile is one. To fit this dataset to the project pipeline, we made adjustments accordingly.

First, the original project has a preprocessing step that resizes the images into 224 x 224 x 3. However, our dataset is in the form of 32 x 32 x 3, which does not require further subsampling for the image classification model. So we eliminate this step. Second, we use SqueezeNet architecture from the MXNet library [9]. This particular architecture is the reason why this application can be run in our edge node, Raspberry Pi, because SqueezeNet requires less than 0.5 MB for the model size [10]. Also note that the model is a pre-trained model of SqueezeNet, which allows us to focus on the deployment of the classification, rather than the training of a new model from scratch.

## 4. Evaluation and Result Metrics

To evaluate the performance of the automobile identification task we measured 4 different statistics, similar to the EdgeBench project. The first measure will be the total compute time. This will strictly indicate the computing power of the Raspberry Pi. Second and the third measure is the propagation and response time from the cloud layer. This will indicate the network speed for the image and the response label to travel in and out of the central repository. The fourth measure is the end-to-end latency. This is simply the summation of all the measures one to three. This measure will be strictly correlated to the quality of service observed by the end users, and therefore, is the key indicator of the system performance. Lastly, we have the payload size as well, but since we are using a standardized image size of 32 pixels by 32 pixels, we observed that all payload sizes were on average the same. These

four different measures will not only be an important identifier for this benchmark but also to compare objectively between the two platforms Azure IoT and AWS Greengrass.

| | Microsoft Azure IoT | AWS Greengrass |
|---|---|---|
| Compute Time (**ms**) | **13.71** | **13.542** |
| Propagation Time (**ms**) | **0.182** | **0.283** |
| Response Time (**ms**) | **0.132** | **0.130** |
| Total End-to-End Time (**ms**) | **14.024** | **13.956** |

**Figure 2. The Performance Measures for each Platform**

Note that the measurements between the two platforms are very similar. The compute time for both platforms is about 13 ms, which is expected since both classification tasks were carried out on a Raspberry Pi. However, it is also surprising to see that the propagation and the response time are similar as well. We would expect that the difference in communication mechanism would cause significant differences. For example, Azure uses a docker container that runs Azure EdgeAgent application to communicate between the device and the cloud, whereas AWS Greengrass uses its own client application running on the Lambda engine. Therefore, both systems appear to be equally efficient in carrying out image classification tasks.

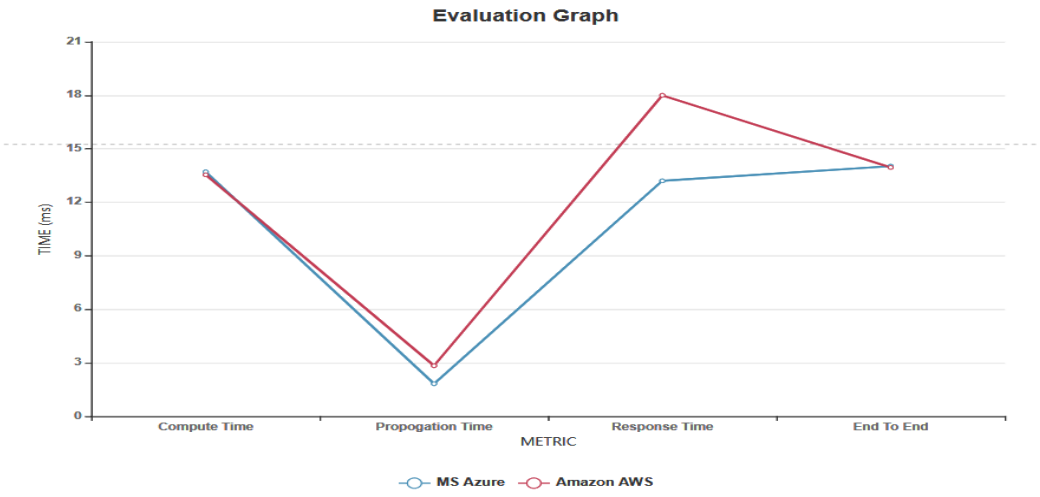


**Figure 3: Comparison of Performance Characteristics between Azure, Greengrass**

## 5. Conclusion and Future Work

In conclusion, we find that both AWS Greengrass and Microsoft Azure IoT are equally efficient in the image classification task on our dataset. Both systems performed very similar results in all three

measurements we collected, and it appears that the communication protocol does not have much bearing on the system performance. For future work, we want to experiment with different architecture configurations in these two environments. One possibility is by introducing a fog layer between the edge Raspberry Pi and the cloud. Another consideration might be to try to relocate the computation to the cloud, and force more communication overhead by requiring to send image files instead of simple classification results.

## 6. References

[1] A. Das, S. Patterson and M. Wittie, "EdgeBench: Benchmarking Edge Computing Platforms," 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Zurich, 2018, pp. 175-180, doi: 10.1109/UCC-Companion.2018.00053.

[2] Barthélemy, J.; Verstaevel, N.; Forehead, H.; Perez, P. Edge-Computing Video Analytics for Real-Time Traffic Monitoring in a Smart City. Sensors 2019, 19, 2048.

[3] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick and D. S. Nikolopoulos, "Challenges and Opportunities in Edge Computing," 2016 IEEE International Conference on Smart Cloud (SmartCloud), New York, NY, 2016, pp. 20-26, doi: 10.1109/SmartCloud.2016.18. [4] R. João et al. 2017. Towards a middleware for mobile edge-cloud applications. In Proceedings of the 2nd Workshop on Middleware for Edge Clouds & Cloudlets (MECC '17). Association for Computing Machinery, New York, NY, USA, Article 1, 1–6. DOI:https://doi.org/10.1145/3152360.3152361

[5] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal and H. Flinck, "Mobile Edge Computing Potential in Making Cities Smarter," in IEEE Communications Magazine, vol. 55, no. 3, pp. 38-43, March 2017, doi: 10.1109/MCOM.2017.1600249CM.

[6] P. Silva, A. Costan and G. Antoniu, "Towards a Methodology for Benchmarking Edge Processing Frameworks," 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Rio de Janeiro, Brazil, 2019, pp. 904-907, doi: 10.1109/IPDPSW.2019.00149.

[7] Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft coco: Common objects in context. In Computer Vision—ECCV 2014, 13th European Conference, Zurich, Switzerland, 6–12 September 2014; European Conference on Computer Vision; Springer: Cham, Switzerland, 2014; pp. 740–755

[8] Russakovsky, O., Deng, J., Su, H. et al. ImageNet Large Scale Visual Recognition Challenge. Int J Comput Vis 115, 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y

[9] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," CoRR, vol. abs/1512.01274, 2015.

[10] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," CoRR, vol. abs/1602.07360, 2016.

[11] G. McGrath and P. R. Brenner, "Serverless Computing: Design, Implementation, and Performance," 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), Atlanta, GA, 2017, pp. 405-410, doi: 10.1109/ICDCSW.2017.36.

[12] Microsoft Azure website. Azure IoT Hub : Managed service to enable bi-directional communication between IoT devices and Azure https://azure.microsoft.com/en-us/services/iot-hub/