Tootiya Giyahchi

Ritwik Nandakumar

Beomjun Aaron Bae

*Benchmarking Fog Network for AWS GreenGrass and MS Azure IoT* : Project Survey Report

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 community, 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 attraction guide videos by local edge storage of location relevant videos [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.

|  |  |
| --- | --- |
| **Fig. 1 EdgeBench GreenGrass Architecture** | **Fig. 2 EdgeBench Azure IoT Edge Architecture** |
| **Fig. 3 EdgeBench Cloud Only Architecture** | **Fig. 4 Our Project Architecture** |

In our project, we will be trying to benchmark the two platforms, AWS Greengrass and Microsoft Azure IoT, by deploying an object recognition application onto a network of multiple Raspberry Pis and measuring baseline statistics like end-to-end latency, resource utilization, and bandwidth usage. Raspberry Pis are chosen because they are often used as a standard for low resource devices. Also, image recognition application was chosen because it is a common task used in neural network testing with significant computational requirements. This setup is intended to analyze the factors that account for system performance in the two widely-used IoT platforms. The remainder of this paper follows the edge benchmarking guideline specified by Silva et al [6].

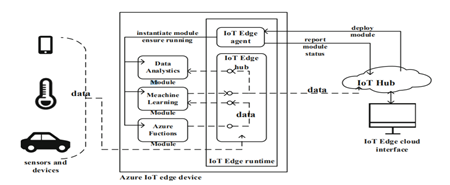
1. Benchmark Objectives

In this project, we intend to extend the existing benchmarking platform, EdgeBench, created by Anirban et al in 2018 [1]. More specifically, we would like to use their open source suite to benchmark a fog layer scheme. 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 scheme is a classical server-client scheme where the end devices send the raw data to the cloud and computation is getting done in the cloud, as shown in Figure 3. However, we intend to explore a third scheme using a fog node.

Simply put, a fog node is a computationally more powerful node than end devices that sits at the edge of the network in close proximity to the end devices. Such a node can improve the performance by enabling more resources for running the model while it decreases the latency and utilizes the bandwidth compared to accessing the resources on the cloud. Figure 4 shows an overview of our proposed architecture. For this project, we will only be focusing on the image recognition application of EdgeBench since Figure 4 architecture is commonly used for such applications that are machine learning based and relevantly heavy so we expect performance gain when we provide a more powerful resource to run the model and, as image recognition applications have a relevantly bigger size of data(images versus messages) therefore having a fog node instead of providing such a resource on the cloud reduces bandwidth occupation and latency. In addition to having a fog node, in our proposed architecture the results are sent back to end devices, very much like architectures found in the industry.

1. Edge Processing Frameworks

Azure IoT Edge is a platform that is an extension to the Azure IoT suite of services. Here, Azure Edge tries to let the analytics be performed at the edge node itself, rather than relying on a central server or cloud endpoint. The platform offers many services like Azure Functions, Azure ML and Stream Analytics, that can be used to compute the IoT task requirements. In an Azure IoT edge setting, we have an IoT Hub (Azure IoT Hub) whose function conforms to its name and is the primary service for IoT that supports cloud messaging related services. It enables message sending between the device to cloud as well as cloud to devices and forms a central point of communication for all other applications. The Azure IoT central and Azure IoT solution accelerators, various services provided by Microsoft as part of the IoT suite, are all to some extent dependent on the IoT hub. Azure IoT Edge consists of three components: IoT Edge modules, IoT Edge runtime and a Cloud-Based Interface. IoT Edge modules are the units of execution in a local device group. Multiple modules can be configured to communicate with each other as well, creating a chain of process execution. In our setup, we will have multiple Raspberry Pi devices working as device nodes in conjunction with our fog (laptop) device. Microsoft allows you to develop custom modules that provide necessary insights offline as well, whose reliability we aim to test through the benchmarking process. The Azure IoT Edge runtime enables custom and cloud logic on IoT Edge devices. The edge runtime has 2 main parts - a combination of an edge agent and edge hub. In Fig. 5, we see an edge agent module present in the runtime environment. This is responsible for retrieving the specified module image, starting up and monitoring the module(s). The edge agent in turn reports the module status to the edge Hub. Usually in an Edge scenario, the code that needs to connect to the IoT Hub is not only running inside a virtualized container, it is also isolated from the IoT Hub by the Edge runtime. This is one of the chief differentiating factors from AWS Greengrass. Azure chooses to implement a virtualization & isolation driven approach with Docker based containers to run workloads. AWS, on the other hand, opts to implement a lambda function /event state-driven approach to perform task completions.. The runtime sits on the IoT Edge device and performs management and communication operations. The runtime can perform several functions like installing & uploading device workloads, report module health, manage to and from device-edge communication, maintenance of edge security, etc. The edge cloud interface serves as a way to manage all your devices in one plane. Workloads for each device can be configured and sent to the appropriate end device (In our case, Raspberry Pi’s), as well as any other device under our control that might be added later. The cloud interface also gives you a way to view and monitor workloads. Azure provides an edge portal UI or device management blade where edge specific functionality will be present, allowing us to monitor each device in real-time.

Fig 5 : Azure IoT Architecture 

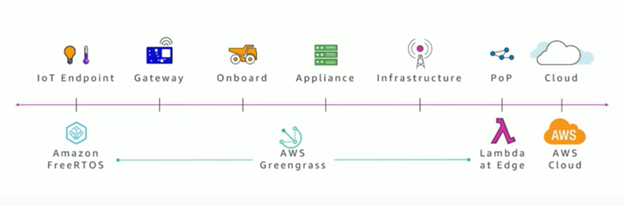
Greengrass IoT, our second benchmarked platform, is an extension to AWS IoT, in that it is an edge compute optimization to the existing AWS IoT suite. Like Azure, there is a central IoT Hub that processes data sent from the devices to the cloud. Before creating a device group on Greengrass, one must define a ‘Greengrass Core’. The core will be the device groups’ local gateway device, the device that runs the AWS Greengrass Core SDK and will allow invocations of lambda functions to work locally on edge devices, as well allow for message execution and interaction with the cloud. Any compatible edge device running the specified Greengrass core software acts as the core. In our case, we will utilize the laptop as a core after which we can create group definitions using the Raspberry Pi devices. The AWS Greengrass Core is said to work locally, even with abrupt changes in connectivity, something that we will look to test and validate in our project. The Greengrass core through the MQTT protocol can handle security aspects like authentication and authorization, as well as message passing to and from the cloud. Once a message is received by the IoT Hub, the data is sent to the cloud, where we can either send it for further processing, send the data to an S3 bucket (or DB) for storage or return it back to the greengrass group itself.

Fig 6 : High Level Overview of the Greengrass Edge Processing Framework

1. Infrastructure of experimental set up

To implement the proposed IoT network in Figure 4, we will use Raspberry Pi 3B model as end devices, and a laptop as the fog node. All devices are connected to the internet via a wireless router. To have them communicate via AWS Greengrass all devices will have AWS IoT SDK installed and will run Greengrass core software so they can be managed and modified through AWS console website. Greengrass authorizes and secures all the communication through MQTT protocol. For Ms Azure IoT, Docker compatible containers will be 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 stick to their configuration as much as possible.

1. Application and Input Data

As mentioned earlier, the application that will be deployed on to our edge system is image classification. Image classification was chosen because the computation load is non-trivial and requires just enough computational power to be carried out anywhere in the system: edge, fog, or cloud. Furthermore, because we want to extend the comparison that Das et al performed on the EdgeBench project, we use the exact same classification pipeline [1].

First, the image is resized into the standard size, which is 224 x 224 x 3, using OpenCV library. Second, we use SqueezeNet architecture from the MXNet library [9]. This particular architecture is the reason why this model is particularly replicable even in a low resource environment like Raspberry Pis because SqueezeNet requires less than 0.5 MB for the model size [10]. This allows us to port the application to edge, fog, and the cloud easily, without significant overhead.

Furthermore, the images that are to be classified are 500 samples from ImageNet dataset [8]. We chose this dataset because EdgeBench library already has a pre-trained model of SqueezeNet that we can implement in our benchmarking [1]. However, the model was trained on the ImageNet dataset to begin with. Therefore, we predict that we want to stay consistent with the original benchmarking procedure in EdgeBench and use the same dataset.

However, there are ways to increase the size of the dataset if we disregard the need to stay true to the original EdgeBench dataset. Obviously, we could sample more from the ImageNet dataset, but also other datasets are available. For example, Barthelmy et al worked on detecting pedestrians and bicyclists using KITTI and COCO dataset [2, 7]. These datasets are slightly different from ImageNet dataset in that the set of category labels are different. Although we predict that the classification will have lower accuracy on these dataset, for the purposes of our project, this shortcoming can be ignored.

1. Evaluation Metrics

To evaluate the performance of our fog computing system, we will be measuring 5 different statistics, just like the EdgeBench project. The first measure will be the total compute time. This will strictly indicate the computing power of the fog node in our network , and it will be measured by the observed computation completion time in the fog node alone. Second and the third measure is the request and the response time from the fog layer. This will indicate the network speed for the image and the response label to travel in and out of different network layers. Fourth measure is the end-to-end latency. This is the summation of all the measures one to three and additional cost in processing the response from the fog layer. 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 will be measuring the payload size and the resource utilization rate such as CPU and RAM. These six different measures will not only be an important identifier for this benchmark but also to compare objectively against the results from the original EdgeBench project.

1. 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] 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.

[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] 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.

[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/