DeFog: Demystifying Fog System Interactions Using Container-based Benchmarking

Jonathan J. McChesney

***Abstract*—Fog computing is an innovative distributed computing model that makes use resources located in the distant Cloud and edge nodes such as routers, at the edge of the network. Many online games, Internet of Things (IoT) and deep learning applications typically make use of the Cloud to deploy applications and handle computation offload. Fog computing proposes moving some of this computation closer to the user and splitting application functionality across the Cloud and Edge to reduce round trip latencies and the cost of communications. Fog benchmarking is an area of benchmarking that focuses on the evaluation of such Fog systems and the resources located at the Edge. This paper proposes an agnostic method for container based benchmarking as a standardised method for the evaluation of the relative performance of fog applications leveraging the Edge, compared to other Edge and Cloud resources. The metrics generated from benchmarking fog systems provide insight into whether the benefits gained from leveraging the Edge outweigh the challenges that manifest with a Cloud-Edge solution. Challenges such as the availability of the limited hardware of edge nodes and the necessity for an uncompromising Quality of Service (QoS).**

***Index Terms*—Fog Benchmarking, Edge Computing, Fog Computing, Cloud Computing, Fog Applications.**

I. Introduction

CLOUD computing only solutions for processing many applications results in latency and abundant communication overheads, which negatively affects Quality of Service (QoS) and Quality of Experience (QoE). Alternative solutions are to bring some of the computation to the edge of the network, closer to user devices. Computing done collectively across the Cloud and resources at the edge of the network is referred to as Fog computing, and applications that benefit from this technique are referred to as fog applications. Fog computing aims to leverage computing at the edge of the network using edge nodes such as routers and base stations. Edge compute resources exist closer to end users and do not incur extensive delays from offloading intensive compute tasks to a distant data center, such as with a Cloud only model [1]. This allows computation offload to be performed closer to the user device; reducing latencies, data traffic and the frequency of communication between the Cloud and user device.

Processing large amounts of data in the Cloud can result in latencies, restrictive costs, and high bandwidth [2]. Latency critical fog applications have real time requirements that can be difficult to maintain with a Cloud only solution [3]. To motivate the decision to leverage the Edge, developers need a way of understanding the relative performance of their application on the Edge in contrast to a Cloud only model. This can be accomplished by benchmarking. Benchmarking is the act of executing a set of repeatable tasks that evaluate the relative performance of an application. Benchmarking tools allow for the evaluation of Cloud and Edge compute resources by generating metrics based on the performance of the application during the benchmark process. Currently in the state of the art, there are no benchmarking methods available for Fog computing. Existing benchmarking tools are for evaluating high-performance clusters, supercomputers, data centers, IoT Edge platforms and Clouds. Such tools and suites are; LINPACK and NAS Parallel Benchmarks which are used to evaluate the performance of high performance supercomputers and DCBench which assesses the performance of data centre workloads. Other benchmark tools include EdgeBench which evaluates Internet of Things (IoT) Edge platforms such as Azure IoT Edge and CloudRank-D which benchmarks and ranks Cloud Computing Systems.

This gap in the state of the art is partly due to the infancy of research regarding fog system benchmarking and the difficulty of generating benchmarks reliably on edge nodes, this is ascribed to the availability of the limited hardware resources of the edge nodes. As such fog system benchmarks will need to be fast in making known compute resource capabilities without compromising the QoS. The aim of this research is to evaluate the differences between fog application performance on the Edge and Cloud, by developing an agnostic benchmarking method for fog applications leveraging the Edge. In this paper a catalog of captured metrics are considered such as communication latency, compute round trip time and the cost of computation. These metrics aim to motivate the benefit of a fog application leveraging the Edge.

The paper proposes DeFog, an agnostic fog system benchmarking method. DeFog executes standardised test scripts on a collection of applications that are distributed across the Cloud and the Edge to obtain metrics from Cloud and Edge compute resources. This is facilitated by using Docker containers that are used for building and deploying the Cloud and Edge components of the Fog application. The use of containerised applications for benchmarking ensures the deployment and benchmarking process is reproducible, reliable and consistent. This is due to the dependencies and build process remaining consistent across the platforms.

In this first edition of DeFog, the following five applications are used for benchmarking: (i) deep learning object classification tool YOLO, (ii) GPS based online mobile game iPokeMon, (iii) speech to text engine PocketSphinx, (iv) forced

J. McChesney is with the School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, UK, e-mail: jmcchesney01@qub.ac.uk

alignment tool Aeneas and (v) IoT Edge gateway application FogLAMP. The research contributions of this paper is the development and use of three distinct deployment pipelines (i) Cloud Only, (ii) Edge Only and (iii) Cloud-Edge combined, to evaluate the feasibility, applicability and benefit of fog applications leveraging the the Edge. Another contribution of this research is the containerisation of fog applications to ensure consistent and fast benchmarks for the evaluation of the fog system. This is achieved by using the proposed generalised fog benchmarking method. This allows a range of workloads to be quickly deployed and benchmarked across the Edge and the Cloud.

The remainder of this paper is organized as follows. Section 2 details the DeFog benchmarking method. Section 3 introduces the application use cases. Section 4 and 5 outlines the system implementation and catalog of metrics. Section 6 outlines the experiments and results. Section 7 presents the related work. Section 8 draws conclusions to this paper.

II. DeFog Benchmarking

To bridge the gap in the state of the art regarding the benchmarking of fog systems, and as a solution to the motivations discussed in this section, this research proposes the generalised fog benchmarking method DeFog.

A. Motivation

Fog system benchmarking needs to consider an application's bespoke requirements [4], for example, the dependencies between the Cloud and Edge components of the application. This in turn makes the set up and execution of fog applications more complex. In Cloud benchmarking, the dependencies of the application are mostly in the Cloud and is therefore not as complex as it is in fog benchmarking.

It is intrinsic for a fog benchmarking method to require the standardised collection of a catalog of metrics on compute resources that are relevant to evaluation of fog applications on the Edge [2]. Such metrics should provide motivation as to whether it is beneficial for an application to leverage the Edge, such as the round trip and end to end latency for a large volume of user devices interacting with the fog system simultaneously [5]. Furthermore, a fog benchmarking method will need to be more agnostic to allow for a large range of application workloads to be evaluated and metrics to be produced. This is in contrast to tools such as DAWNbench which are focused on evaluating a specific type of workload [6].

The Edge is a dynamic and versatile system, but with the limited availability of the hardware of the edge nodes, benchmark results will need to be generated quickly [4] [7], ideally within a matter of minutes. This can be achieved by executing a benchmark run for each individual workload provided as input to the containerised application. The proposed method has been evaluated to complete a benchmark run in under one minute on average, for all fog applications leveraging the Edge. In addition to this, a fog benchmarking method will need to generate benchmark metrics on both the Edge and Cloud solutions to capture comparable and motivational results. The consistency of the benchmarks is an important consideration that will need to be preserved on both the Cloud and Edge to ensure similar and 'fair' benchmarks are performed each time. This can be achieved by containerising applications to ensure the same application build version and package dependencies are consistently deployed and tested across the Cloud-Edge models for each benchmark run.

Many benchmark tools only consider the metrics relating to a specific workload such as Deep Learning End to End latency and inference [6], a general fog benchmarking method will need to generate metrics for a much larger range of application workload types that can be deployed to the Edge. Furthermore, different applications prioritise different metrics, and there is no current method for reliably generating these metrics on the Edge. This unreliability is partly due to the challenge of ensuring the edge nodes achieve a high throughput and are not overloaded with computationally heavy workloads [5]. This research method proposes reliably capturing metrics by reducing the platform noise during the benchmark process [5]. This is achieved by executing the benchmarking process multiple times for a range of input asset data using separate but identical containers for processing each workload.

Fog applications that run on the Edge have real time requirements that are difficult to be met by the geographically distant Cloud [2]. Additional communication is required between the Cloud and Edge systems to fully represent the Cloud-Edge model to generate feasible application benchmark metrics. Such applications include online games which are greatly affected by communication latency and the resulting screen delay [8]. By evaluating communication latency metrics and successful response rates attributes, insight can be obtained for ensuring a good QoS and QoE [9].

B. DeFog Architecture

To address the motivations presented above and to further explore how a developer can gain value from evaluating applications deployed across the Cloud and the Edge, DeFog will be explored within this section. The proposed DeFog method uses a 7 step methodology for each benchmark iteration that deploys, runs and returns the catalog of application metrics. These steps are (1) build the application image on the destination platforms, (2) run a detached container of the application, (3) transfer an asset payload to the destination instance e.g. an audio file to be converted into text, (4) Transfer model asset to the Edge instance e.g. an acoustic language model, (5) run computational task, (6) Generate catalog of metrics based on captured attributes, (7) Return metrics to user. Step 4 is only necessary for the Cloud-Edge pipeline. Step 3 to 6 is repeated several times, as a single benchmark run occurs for each asset. Additionally, DeFog proposes benchmarking the platform itself to generate attributes that are beneficial to evaluating Cloud-Edge comparative performance.

Step 1) Build application image

The selected application image is built and tagged, ensuring a consistent environment for all future benchmark runs.

Step 2) Run application container

The application container is run in detached mode, allowing for concurrent benchmark tasks to be executed for separate but identical application containers.

Step 3) Transfer data workload

A workload such as an image to be classified or simulated player behaviour data is transferred to the platform service. This workload varies for each application. For example, the asset folder contains X assets, where $X = \{ x_1, x_2, ..., x_d \}$, and d is the total number of assets. For each successive benchmark run the next asset x_i is transferred to the destination platform, until all workloads have been transferred. Communication metrics are also generated at this step.

Step 4) Transfer model asset

The Cloud-Edge pipeline leverages communication between the Cloud and Edge by transferring a pre-trained model asset to the Edge. This simulates the communication between the global state storage of the Cloud [7] and the more versatile Edge. The Cloud is assumed to have performed the computationally intensive task of training the models and weights. Further communication metrics are captured at this step.

Step 5) Execute compute task

For application Y, a computational task that results in the generation of data output is executed, such as creation of a synchronised mapping file that correlates an audio segment to a chunk of text. Computational metrics and the real time factor for this task are generated and then returned to the user device. For example, application Y is run d times, where d is the number of assets workloads provided to the containers.

Step 6) Compute metrics catalog

The attributes captured from the previous steps are used to compute a catalog of metrics that provide actionable insights towards exploring the feasibility and benefit of the fog application leveraging the Edge. For example, There are n potential metrics that DeFog will consider. These metrics can be grouped into three categories (a) communication metrics such as return trip latency and data transfer rates, (b) computation metrics such as the time taken for a computation task and the real time factor value of a speech to to text fog application. Finally DeFog considers a third category (c) concurrency metrics, such as latency and response success rates for concurrent users. For example, benchmarking application Y may generate a subset of n potential metrics m, (where, $1<=m<=n$). This provides flexibility to the catalog of metrics as some applications may prioritise certain attributes over other, or consider application specific attributes such as the real time factor for speech to text applications.

Step 7) Return metrics to user device

The attributes are returned to the user device after X workload benchmarks have finished. These metrics are parsed into a series of file types that provide actionable insights to the user. For example, m metrics are converted into a .csv file, as well as a verbose text file and are saved on the user device.

C. DeFog Pipelines:

The architecture proposed for DeFog works across three distinct pipelines as shown in figure 1.

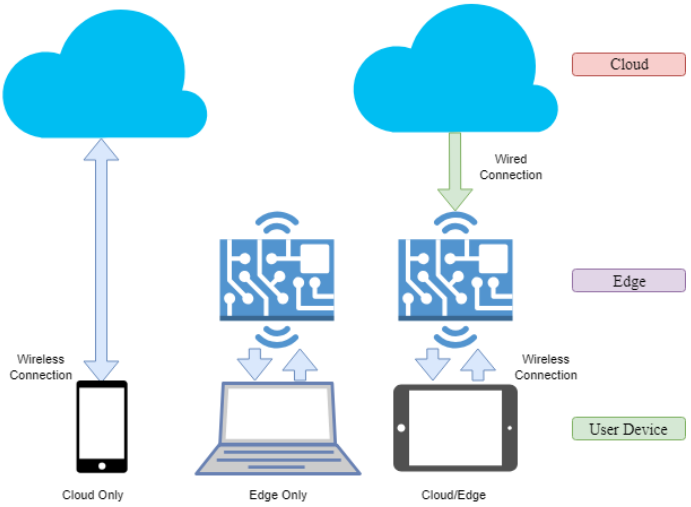


Fig. 1: Defog Deployment Pipelines.

The first pipeline as shown in figure 2, is the Cloud only pipeline, which consists of two tiers: the Cloud instance and the user device. The Cloud tier is where the application is built and deployed using docker containers. A workload is provided to the containerised application, next a workload specific computational task is executed and benchmark metrics are generated. The user device tier concerns the myriad of diverse gadgets, from smartphones, wearables and laptops [10]. The user device tier uses Secure Shell to interact with the Cloud tier during the benchmarking process.

A connection is established between the user device and the Cloud instance, where application Y is built and deployed using docker. The user specifies the fog application to be benchmarked and an asset workload is transferred to the Cloud instance. Within the containerised application a computational task is executed on the Cloud platform, and the data output such as a classified and labelled image is uploaded to a Simple Secure Storage (S3) bucket. This is also transferred back to the user device. Metrics are generated throughout the benchmarking process and are also returned to the user device.

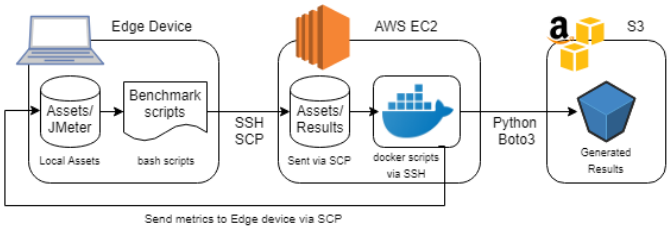


Fig. 2: Cloud Pipeline using AWS.

To enable the evaluation of the Edge, the Edge only pipeline is offered, as outlined in figure 3. This consists of a similar tier structure: the Edge and the user device. The Cloud tier is substituted for the Edge Tier, as this concerns the edge nodes that are geographically closer to the user and perform the

necessary computations. In contrast to the Cloud only model where applications leverage traffic routing nodes [7], the Edge model makes use of the computational capabilities of these edge nodes. Odroid XU4 and Raspberry Pi 3 boards are used to simulate the hardware availability of edge nodes.

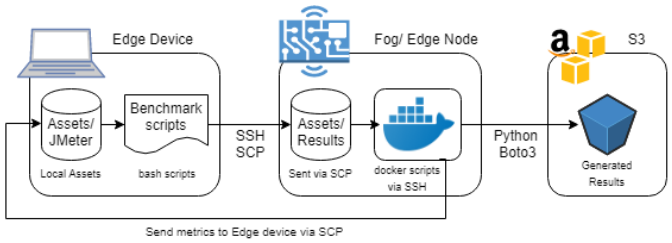


Fig. 3: Edge Pipeline using an OdroidXU4/ Raspberry Pi 3.

The Cloud-Edge pipeline, as shown in figure 4, leverages the geographically closer and more versatile Edge platform [5], with the more computationally powerful hardware of the Cloud used to extend functionality [10]. This solution leverages the Clouds superior hardware and the Edge's close proximity by allowing for a communication stage. This mitigates potential extensive computations on the Edge regarding the training of weights and model data.

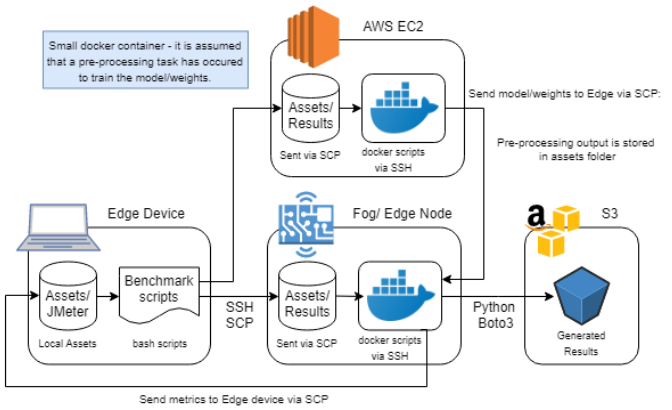


Fig. 4: Cloud-Edge (Fog) Pipeline using an AWS and OdroidXU4/ Raspberry Pi 3.

For example, a user builds and deploys a deep learning classifier application Y to the Edge using docker and transfers to the edge node an image asset x_1 to be classified. The Cloud instance then transfers a pre-trained (AlexNet) weights model to the Edge for an inference task [11]. The Edge uses the weights file to run a detection and classification task on the provided image. The classified image and captured metrics are then returned to the user device and uploaded to an S3 bucket. As this method emulates the behaviour and hardware capabilities of edge nodes using the Odroid and Raspberry Pi boards, an assumption is made that the training and generation of the model and weights files is performed in the Cloud before the benchmarking process begins.

D. Cloud Platform: Amazon Web Services

Amazon Web Services (AWS) is the cloud platform explored in this research. Cloud benchmarking is performed using an Elastic Cloud Compute (EC2) instance running Ubuntu.18.04. Results data is stored in an Simple Secure Storage (S3) bucket and also transferred to the user device, as shown in Figure 2. The AWS command line interface (CLI) tool is used to set Information Access Management (IAM) credentials for remote access of the Cloud instance. Secure Shell (SSH) is used to ensure a secure connection is maintained with the Cloud.

Typically the Cloud is leveraged as a large scale data centre that can handle a large volume of user traffic by leveraging load balancers and routers [12]. This large scale Cloud centralised model results in a large increase in the frequency of communication [5], this incurs latencies, high bandwidth and network bottlenecks [2]. While network communication exists as a bottleneck in the Cloud, the computational performance of intensive workloads excels due to the powerful hardware available to be leveraged in the Cloud [10].

E. Edge Nodes: Odroid XU4 and Raspberry Pi 3

Edge nodes such as routers have limited availability of hardware in contrast to the Cloud, but exist geographically closer to the user. This mitigates some of the network performance bottlenecks of a Cloud only solution [2]. The close proximity of edge nodes is particularly beneficial for latency critical applications that require real time responses [5]. This solution evaluates the communication, computation and concurrency metrics generated using the Edge in contrast to a Cloud Only solution.

With the emergence of Fog computing many companies are exploring the benefits of the Edge as a platform, such as Google's Cloud IoT Edge, Microsoft Azure IoT Edge, IBM Watson IoT Platform Edge and AWS Greengrass [2]. Leveraging compute nodes in relative close proximity to the data source to perform data processing, game state updates and object classification reduces application network response time and communication latencies [13]. This research evaluates the Cloud-Edge model to gleam insight into the performance benefits gained for different fog applications leveraging the edge.

III. Applications Used in DeFog

In this section, the Fog applications benchmarked within DeFog are outlined.

TABLE I: Fog Applications used in DeFog.

Application	Description	Application Type
YOLOv3	Object Classification	Deep Learning
PocketSphinx	Text to Speech Convertor	Text to Speech Engine
Aeneas	Text-Audio Synchronisation	Forced Alignment
iPokeMon	Online Mobile Game.	Latency Critical
FogLAMP	IoT Edge Gateway App	IoT Application

Currently 5 applications are used to generate workloads for evaluation using the DeFog benchmark method, as shown in table 1. With the expansive growth of mobile gaming technology, along with massive advancements in deep learning image classification applications [14], the Cloud is typically leveraged to extend the capabilities of edge devices [10].

With the emergence of Fog computing [15], edge nodes while operating on a smaller scale from the Cloud, reside closer to the user device. For this reason, it is interesting to evaluate the relative communication and computational performance of Fog applications leveraging the Edge, and compare with a Cloud Only solution to gain actionable insights.

A. Object Classification (YOLOv3)

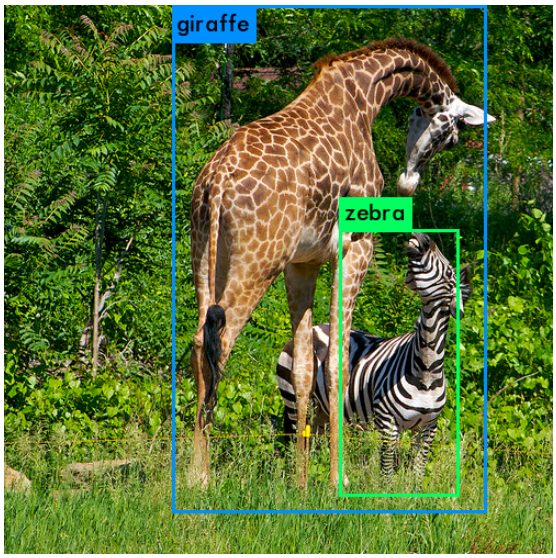


Fig. 5: YOLOv3 Sample Predicted Output.

[1] This is a fork of the You Only Look Once (YOLO) deep learning object classification tool [16]. This open-source project runs on Windows and Ubuntu platforms, and unlike the main Yolo application codes is compatible with Raspberry Pi's and other boards. This uses Darknet's Yolov3 and Yolov2 object classifier, and utilises AlexNet weights. A smaller set of pre-trained weights is used, this results in faster computation at the expense of prediction accuracy. The computational task executed in the experiments is the TOLOv3 detector test function, this re-sizes a provided image asset and performs comparisons to detect what objects are present within the image. A labelled image is generated with percentage weights defining the threshold accuracy output printed to the console, example output can be seen in figure 5. Various image assets are transferred to the application during pipeline benchmarking, with the computational task executed on each image generating a unique predicted output image. This predicted image is transferred to an S3 bucket and back to the user device.

B. Speech to Text Convertor (PocketSphinx)

[2] An open-source large vocabulary, speaker-independent continuous speech recognition mobile engine developed by Carnegie Mellon University. PocketSphinx converts a supplied .wav file to a defined language in text form, the experiments make use of an eu-us pre-trained acoustic model to determine the source and destination language for speech to text conversion. The integrated assets are sourced from the large scale speech repository. [3]. The function tested in these experiments is the continuous input function. This accepts an audio file input instead of direct speech input. The supplied audio asset workload is detected and parsed using the acoustic model and then converted to text. Modifications have been made to the application to save the verbose text output to file. The edge device supplies a .wav asset workload to the containerised instance. The acoustic model file is then transferred to the Edge if running the Cloud-Edge pipeline. The computational task is then run for the workload, and a text file is generated for the converted text. This text data is then uploaded to the S3 bucket as well as the user device. Computation and communication metrics are captured throughout this process.

C. Forced Aligment (Aeneas)

[4] This is a library comprised of a set of tools to automatically synchronize text and audio segments. This tool generates synchronization maps (.smil) for a list of text fragments and an audio file containing the relevant text, this is referred to as forced alignment, as shown in figure 6. Aeneas determines a mapping between corresponding audio segment for each text supplied. The user device transfers an audio (.wav) file and a text segment (.xhtml) to the container instance for benchmarking. If running the Cloud-Edge pipeline, only the .wav file is supplied with the text segmentation occurring on the Cloud before the text segment is transferred to the Edge. Forced alignment is performed on the supplied workloads, and the resulting synchronisation map is transferred to the S3 bucket and user device, along with the catalog of generated metrics.

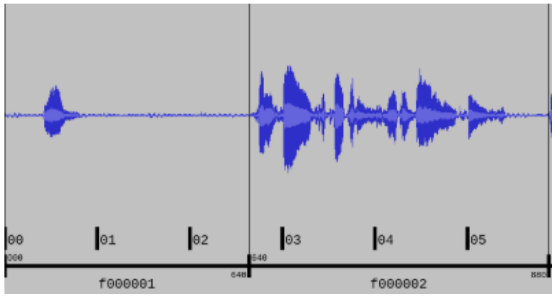


Fig. 6: Aeneas Synchronisation Map.

D. GPS Mobile VR Game (iPokeMon)

[5] This is an open-source location based virtual eality online mobile game. The user starts the iPokemon server and then uses JMeter and Taurus to simulate player behaviour by generating synthetic workloads [9]. The user device supplies the .jmx file to JMeter, this allows for the automated generation of workloads for a range of concurrent users. The captured metrics are then returned to the user device.

[1] https://github.com/AlexeyAB/darknet
[2] https://github.com/cmusphinx/pocketsphinx
[3] http://www.repository.voxforge1.org/downloads/SpeechCorpus
[4] https://github.com/readbeyond/aeneas
[5] https://github.com/qub-blesson/ENORM

E. IoT Fog Application (FogLAMP)

[6] This is an open source Internet of Things (IoT) Edge gateway application that integrates Cloud storage and senors. FogLAMP is a service that when running can interact with endpoint sensors or itself to collect and aggregate statistics and data. The user transfers to the container instance a text payload containing a curl command. The running service executes this curl command to invoke a mocked API call, that gathers localised data. The text results of this are "beautified" and formatted before being transferred to the S3 bucket and user device, along with the catalog of metrics.

IV. Implementation

A. Infrastructure

GitLab [7] and GitHub [8] are used to manage and store the DeFog codebase. This repository also contains the modified fog applications that are benchmarked using DeFog. DeFog uses the repository path and the various sub-trees to pull application specific Dockerfiles for building application images and containers. The DeFog system is written in bash and python and uses scripts and Dockerfiles to run benchmarks.

An AWS Elastic Compute Cloud (EC2) instance is set up in the Dublin, Ireland eu-west-1 region. An Identity and Access Management (IAM) user is instantiated, and public and private keys are configured to allow for secure remote access and control of the EC2 instance. The Cloud is allocated a 16Gb Volume to ensure adequate space is available for building several distinct fog application containers simultaneously. The latest Docker dependencies are installed onto the instance to allow seamless container deployment.

The Odroid XU4 and Raspberry Pi 3 boards are set up with Ubuntu 14.04 and the latest version of Raspbian respectively. The edge device running these benchmarks has the following specifications: Intel i5-6300HQ 2.3GHz CPU, 8GB ram and dedicated GTX 960 GPU.

Docker 17.12.1-ce is used to automate fog application deployment, building and compute task execution. Each fog application's unique Dockerfile installs the relevant dependencies and packages during the build process. Container instances are then run using the same application image.

Each pipeline follows a similar workflow, as shown in figure 7. For each benchmark a data payload is transferred from the user device to the Edge or Cloud, with the Cloud-Edge pipeline requiring a model asset from the Cloud to be used as an additional input on the Edge. A computational task is performed on the destination platform service and the generated data is then transferred to the S3 bucket and user device. Finally metric data is computed, returned and parsed.

Yolo, PocketSphinx and Aeneas are benchmarked across all three pipelines, while FogLamp due to the nature of this IoT application and difficulty splitting its workflow across the Cloud and Edge, doesn't require a Cloud asset, and so is not run for the Cloud-Edge pipeline. These benchmarks are executed several times and then averaged to reduce outliers.

[6]https://github.com/foglamp/FogLAMP
[7]https://gitlab.eeecs.qub.ac.uk/40126401/csc4006-EdgeBenchmarking.git
[8]https://github.com/qub-blesson/DeFog

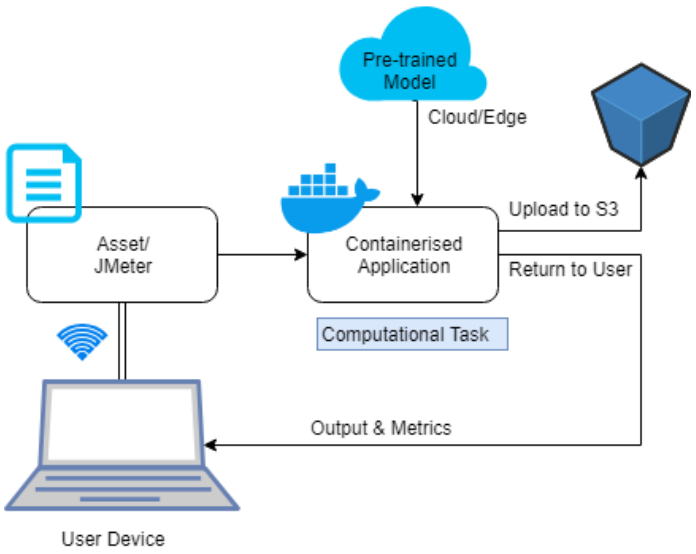


Fig. 7: Typical pipeline workflow.

B. Cloud Only Deployment

Several distinct application workloads for benchmarking can be performed solely on the AWS Cloud to process data and generate results. For latency critical workloads such as YOLO, Aeneas, FogLAMP and PocketSphinx, the user device communicates only with the Cloud instance. The Cloud transfers the results data back to the user device, and the S3 bucket.

The Cloud Only pipeline for each fog application is run via the command line using DeFog on an edge device, this process uses Secure Shell (SSH) and Secure Copy Protocol (SCP) to interact with the Cloud Service. Depending on the fog application selected for benchmarking, automated scripts pull the respective docker files from the DeFog repository. A sequence of scripts are autonomously invoked to build the fog application image and container. An application container will be instantiated for each asset payload. For example, in the YOLO use case, an image asset is transferred to the server from the edge device using SCP. SSH is then used with the Cloud instance to execute the containerised fog application's computational task to be benchmarked, and calculates the compute latency metrics. The predicted image results are transferred to both an AWS S3 bucket and the edge device.

C. Edge Only Deployment

The Edge pipeline is also run via the command line by specifying the edge node instance to leverage. SSH credentials are setup externally to allow for automated authentication between the Odroid XU4 board, Raspberry Pi 3 and the edge device. Similar to the Cloud Only deployment pipeline, docker is used to build and run each fog application. The results data is transferred to both the S3 bucket and user device, and computed metrics are returned to the edge device to be used in the generation of the remainder of the metric catalog.

D. Cloud-Edge Deployment

The Cloud-Edge pipeline deploys the fog application across both the Edge and Cloud platform services, as such this

includes a communication layer between the Edge and Cloud instances not present in the other pipelines for resource offloading [17]. Docker is used to build and deploy the Cloud and Edge application containers. The Cloud instance contains a model or weights asset that is used as input on the Edge. The Edge instance performs a compute task that requires a model asset to be transferred from the Cloud and used as input. In the case of YOLO, this is the pre-trained AlexNet weights. This model asset is transferred from the Cloud to the Edge accordingly. This proposed research makes an assumption regarding the computational task that is performed on the Cloud instance, as it is assumed the compute intensive task of training the model or weights file has already been performed on the Cloud before the execution of the benchmark pipeline.

V. Benchmark Metrics:

Benchmark tools such as cheetah sst-benchmark[9] and Wakeword-benchmark[10] offer insight into the computational performance of specific workloads pertaining to speech to text applications. Similarly TailBench[11] benchmarks the "tail latency" of a selection of latency critical applications, but does not explore other workloads, performance metrics or Cloud and Edge motivations [18]. The proposed Fog benchmarking method on the other hand allows for a range of fog applications and workloads to be benchmarked on both Cloud and Edge platforms. This provides a catalog of relevant metrics that evaluate the relative computational and network performance of both the platform and fog application.

EdgeBench[12] is an IoT Edge benchmarking tool that evaluates IoT edge services such as AWS Greengrass, as opposed to benchmarking the edge nodes themselves, which is intrinsic of a Fog benchmarking method for containerised applications [2]. Hyperfine[13] and nench[14] are benchmarking tools that evaluate I/O and network performance by utilising the command line user interface. Nench and a similar tool bench-sh[15] use bash scripts to execute command line system benchmark. DeFog similarly uses bash and the command line to provide a simplistic yet powerful system and user interface for interacting and performing comprehensive benchmarks.

A. Platform Metrics

In the state of the art, tools such as Phoronix[16] an open-source, cross-platform benchmarking suite and iPerf[17] provide network and bandwidth test services to evaluate platform resources. The benchmarking method proposed in this research aims to bring comprehensive platform metrics to the Edge. This is achieved by benchmarking the computational and network performance of both platforms, and by integrating tools such as Unixbench and Sysbench. Currently in the state of the art there is no comprehensive method that benchmarks both the Cloud and edge nodes as well as the fog applications running on these services. DeFog aims to bridge this gap by benchmarking the platform metrics as shown in table 2.

TABLE II: DeFog Platform Metrics.

Name	Short Name
CPU Model Name	The model name of the platform's CPU.
Number of Cores	The total cores available on the CPU.
CPU Frequency	The frequency of the CPU.
System Uptime	The total time the system has been running.
Unzip time	The total time taken to unzip a 34Mb weights file.
Download rate	The download rate (MB/s) of a 200Mb model file.
System I/O	Speed of I/O reading and writing.

In addition to providing insight into the performance of the Cloud service and Edge nodes, the proposed method uniquely utilises automation and Docker to streamline the deployment and benchmarking process on both the Edge and the Cloud. This allows for fog applications to be built and benchmarked on the versatile Edge within minutes [4].

B. Fog Application Metrics

DeFog provides insight into a range of workload types for fog applications such as Yolo, PocketSphinx, Aeneas and FogLamp. The metrics as outlined in table 3 provide actionable insights to a user regarding the fog applications relative performance on the Cloud or Edge. This informs whether it is beneficial to leverage the Edge. The proposed method prioritises the benchmarking of three categories. 1) Communications, i.e. The overhead of transferring assets and data payloads during an applications execution. 2) Computational performance, i.e. the return trip time including the time taken to execute a computational task. 3) Concurrency, i.e. the impact on the generated metrics from background platform noise and multiple simultaneous benchmarks.

C. Metrics from External Tools

The Edge aims to meet the real time requirements of latency-sensitive applications [1], such as VR based online mobile games by bringing compute and network resources closer to user devices [19]. This is often achieved through leveraging peering edge servers [20], while maintaining a global game state in the Cloud service [7] [8]. End to end latency, throughput and goodput are important metrics for applications that require consistent and frequent communication output. As such, latency and throughout results can be gained from the metrics provided by using JMeter [18] to simulate client player behaviour, as shown by table 4. If a high latency is observed, this would imply the Fog system does not perform well for the quantity of concurrent users. Taurus[19] is used to gather more detailed response metrics such as successful response count and average response time, as observed in table 5. By simulating synthetic user workloads [9], beneficial metrics such as the standard deviation of the response time can be generated, this offers insight into the effect of outlier responses on the median response time.

[9]https://github.com/Picovoice/stt-benchmark
[10]https://github.com/Picovoice/wakeword-benchmark
[11]https://github.com/lyuhao/tailbench
[12]https://github.com/akaanirban/edgebench
[13]https://github.com/sharkdp/hyperfine
[14]https://github.com/n-st/nench
[15]https://github.com/hidden-refuge/bench-sh-2
[16]https://github.com/phoronix-test-suite/phoronix-test-suite
[17]https://github.com/esnet/iperf
[18]https://jmeter.apache.org/
[19]https://gettaurus.org/

TABLE III: DeFog Fog Application Metrics.

Name	Short Name	Description
Execution Time	ET	The time taken to complete a computational task.
Time in Flight	T1	The time taken to transfer a workload to the Cloud or Edge.
S3 Transfer Time	T2	The time taken to upload results data to the S3 bucket.
Results Transfer Time	T3	The time taken to return results data to the user device.
Computation Latency	RTT	Total time taken for the benchmark process including the execution time (RTT=T1+ET+T3).
Computation Cost	Cost	The estimated cost of the computational task (using the AWS pricing strategy).
Real Time Factor	RTF	The rate of speech recognition (ET / file length).
Bytes Up	BytesUp1	The total bytes transferred to the Cloud or Edge.
Bytes Down	BytesDown1	The total bytes transferred from the Cloud or Edge.
Bytes Down (Cloud to Edge)	BytesDown2	The total bytes transferred to the Edge from the Cloud when transferring the model asset.
Bytes Up Per Sec	BytesUp/s1	The bytes per second upload rate.
Bytes Down Per Sec	BytesDown/s1	The bytes per second download rate.
Bytes Down (Cloud to Edge) Per Sec	BytesDown/s2	The bytes per second download rate for transferring the model asset to the Edge.
Cloud/Edge Transfer Time	T4	The time taken to transfer a model or weights file from the Cloud to the Edge.
Communication Latency	CL	The total time taken for payloads to be transferred throughout the process (CL=T1+T3)
Full Computation Latency	FRTT	The return trip time including the time taken to transfer the Cloud model to the Edge.
Full Communication Latency	FCL	The total time taken for the transfer of all payloads including the Cloud model asset.

TABLE IV: JMeter Metrics.

Name	Description
Completion	Success or Failure response of the supplied workload.
User/ Thread	The total concurrent users/threads.
Latency	Response time latency for a specific endpoint.

TABLE V: Taurus Metrics.

Name	Description
Concurrency	Average number of concurrent users.
Throughput	The total count of sample workloads.
Success/Fail	The total count of successful workloads.
Average Response Time	Average response time of service.
Standard Deviation RT	Standard deviation of the response time.
Average Latency	Average latency time for the return trip.

VI. Experimental Studies

The experimental setup for capturing the metrics of several fog application use cases is presented in this section.

Currently benchmark suites such as GooglePerfKitBenchmarker[20] integrate a plethora of third party cross platform benchmarking tools to provide a larger range of evaluative metrics for different workloads. A fog benchmarking method would benefit from this by integrating services such as UnixBench[21] and SysBench[22] to provide insight into the the Cloud or Edge platform service running the fog application. The computational and network performance of the Edge is evaluated and comparisons can then be made with the Cloud instance benchmark results [10]. As such, Unixbench and Sysbench are integrated within DeFog, to generate comprehensive CPU, concurrency and I/O read write platform metrics. This provides insightful value to a user regarding whether the Edge platform is feasible, even before deploying.

Fog computing has the potential to minimise the cost, bandwidth and latency bottlenecks [2] that occur due to the high latency and restrictive costs of using a Cloud Only solution [21]. DeFog allows for data transfer rates, real time factor values, computational costs, as well as end to end communication and computation latency metrics to be generated quickly [4]. The Edge while more versatile than the Cloud has limited availability of hardware [7], as such the three pipelines are used to comparatively evaluate a range of workloads. A subset of the metrics generated are evaluated in this section, such as the average of all the captured bytes transferred per second attributes, as well as the communication and computational latency metrics. These are the sum of several metrics such as the time in flight, execution time and results return time, as shown in table 3. The most relevant external tool metrics such as the response latency, response success rate and the standard deviation results are also evaluated in this section.

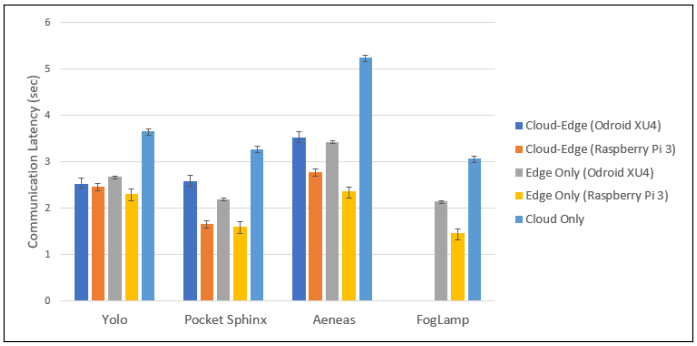
A. Setup

DeFog's infrastructure is set up as outlined in section 2. A series of experiments are run for each fog application across the three pipelines. The aim of each of these experiments is to compare and evaluate the relative computational and network performance when leveraging Cloud or Edge compute resources. Further experiments are performed to evaluate the platform service and fog application performance in a noisy environment [5], which is reflective of real world use.

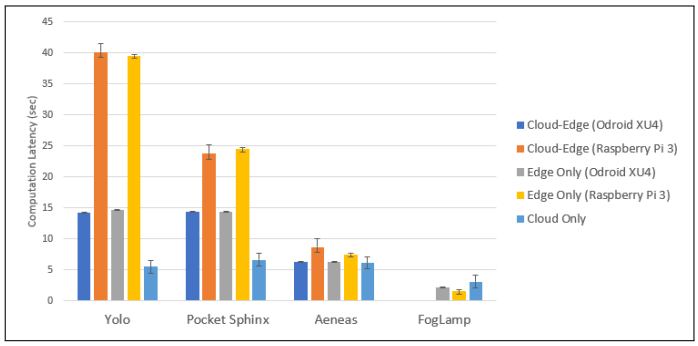
B. Experiments

The Edge is a more versatile system when compared to the Cloud, DeFog leverages this by benchmarking both the platform and fog application compute resources within seconds [4]. Platform benchmark metrics are captured for both services to evaluate the relative network and compute performance. The Edge is limited by the availability of hardware [7], but the edge nodes reside closer to the user than the geographically distant Cloud [10]. This should be reflected in the results, as the Cloud should resolve faster computational performance metrics, but the Edge should show benefit from reduced network delays and latencies. The aim of these benchmarks is to evaluate different workloads and determine if certain fog applications can benefit from leveraging the Edge. The goal of these experiments is therefore to evaluate the state of the art pertaining to fog systems benchmarking on the Cloud and Edge.

[20]https://github.com/GoogleCloudPlatform/PerfKitBenchmarker
[21]https://github.com/kdlucas/byte-unixbench/tree/master/UnixBench
[22]https://github.com/akopytov/sysbench

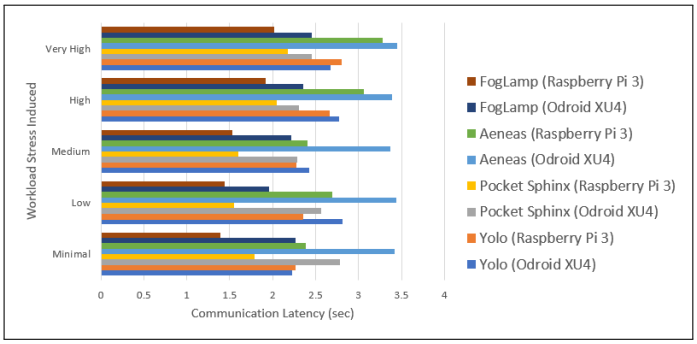


(a) Communications Latency for the Cloud Only and Cloud-Edge pipelines for K use-cases

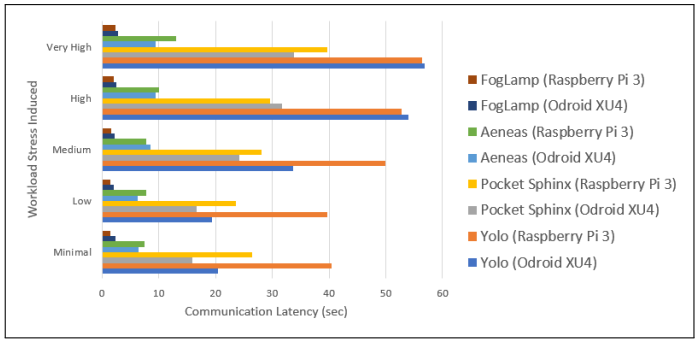


(b) Computation Latency for the Cloud-only and Cloud-Edge pipelines for K use-cases

Fig. 8: Fog Application Benchmarks without Platform Noise:

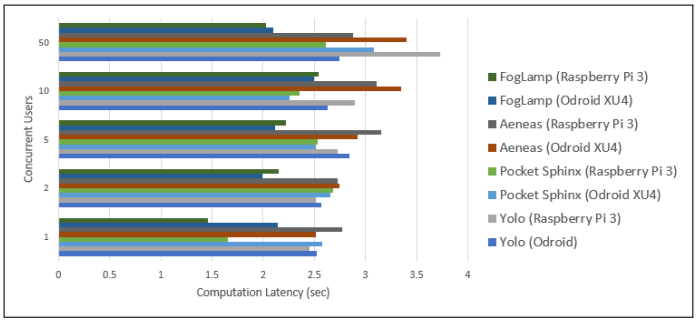


(a) Communication latency for the Cloud-Edge for K use cases for D network intensive data loads.

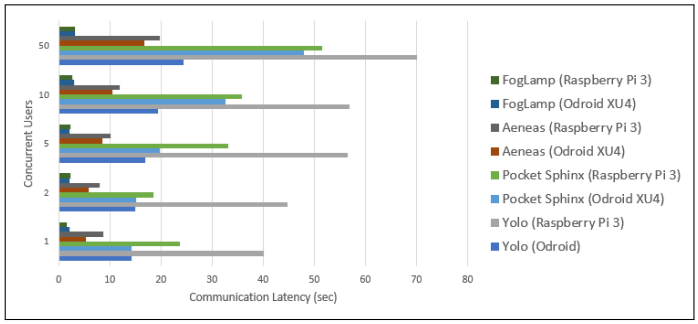


(b) Computation latency for the Cloud-Edge for K use cases for D compute intensive data loads.

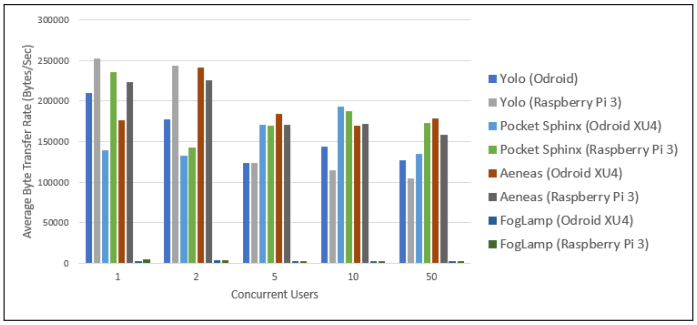
Fig. 9: Fog Application Benchmarks with Induced Gaussian Workloads:



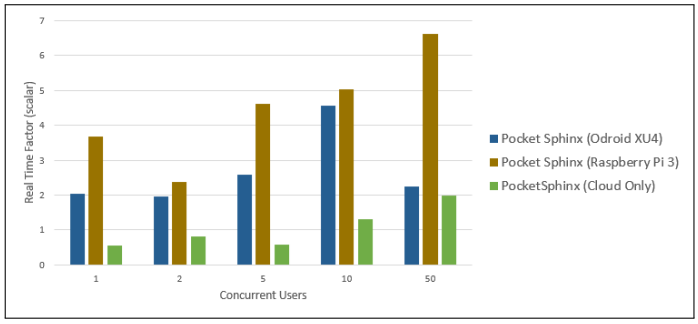
(a) Communications Latency on the Cloud-Edge pipeline for K use cases for N Concurrent Users.



(b) Computation Latency on the Cloud-Edge pipeline for K use cases for N Concurrent Users.

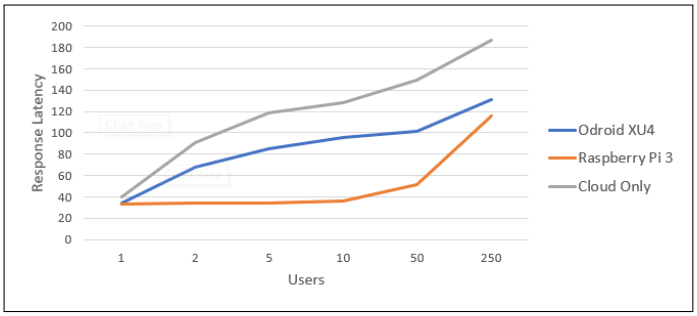


(c) Data Transfer Rate averaged across Upload and Download on the Cloud-Edge pipeline for K use cases for N Concurrent Users.

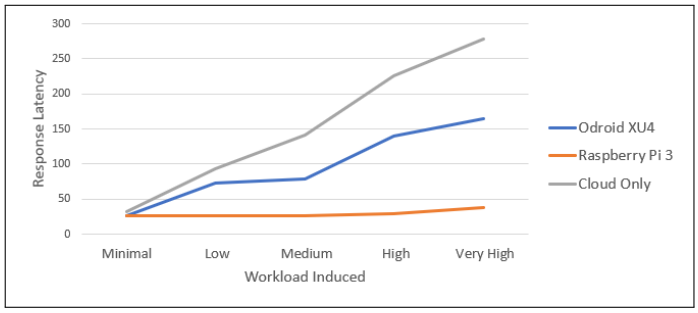


(d) Real Time Factor value for PocketSphinx on all pipelines for K use cases for N Concurrent Users.

Fig. 10: Fog Application Benchmarks For Concurrent Users:



(a) Response Latency for N concurrent Users.



(b) Response Latency for D network Gaussian stressors for 10 Users.

Fig. 11: iPokeMon Simulated User Response Metrics on the Edge and Cloud Only Pipelines.

TABLE VI: Platform Benchmark Results.

Metric	**Amazon Web Services - Elastic Compute Cloud**	**Odroid XU4**	**Raspberry Pi 3**
CPU Model Name	Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz	ARMv7 Processor rev 3 (v7l)	ARMv7 Processor rev 4 (v7l)
Number of Cores	1	8	4
CPU Frequency	2399.969 MHz	2000.0000	1200.0000
Unzip time	3.74 secs	600.912130005 secs	540.654467249 secs
Download rate	9.97Mb/s	404 KB/s	449 Kb/s
System I/O	67.8 MB/s	4.4Mb/s	8.5Mb/s

Experiment 1: Platform Benchmarks & Observations:

To fully evaluate each fog applications performance on the Cloud and Edge the platform metrics outlined in table 2, are evaluated in table 6. These results offer insight into the capabilities of the hardware running the fog applications.

It's interesting to note the large disparity between the Cloud and the Edge platform metrics. The Cloud while having less cores, has a greater CPU frequency and considerably faster I/O rate and network speed. Regarding the slow network for the Edge, this is potentially due to the slow network speed of the test environment used to benchmark the edge nodes. It can be deduced that the Cloud platform has far superior computational hardware than the edge nodes. A difference can also be observed between the edge nodes capabilities, as the Raspberry Pi 3 benchmarked has a newer CPU model, less cores and a lower CPU frequency than the Odroid XU4 board.

Experiment 2: Fog Application Results & Observations:

In this experiment the fog applications are benchmarked without any noise, such as external systems interacting with the platform. This experiment focuses on the computation and communication round trip times presented in figure 8. The standard deviation is presented for all data columns, to highlight the consistency of the captured metrics, conversely a high standard deviation indicates inconsistency and outliers.

A shown in figure 8 (a), the communications latency is consistently lower for all applications running on the Edge compared to the Cloud. It is also worth noting there is a slight disparity between the latencies computed on the edge nodes, with the Odroid XU4 benchmarks having slightly larger latencies. The lower latency of the Edge is due to the closer proximity of the edge nodes, even though the Cloud has a much faster network speed as evaluated in experiment 1. The variance in the edge nodes results is also due to the disparity in device hardware as observed in experiment 1. These results are consistent for all communication metrics captured as detailed in table 3, such as the time in flight and results transfer time.

The compute round trip time as observed in figure 8 (b) is significantly larger for computationally intensive applications such as Yolo and PocketSphinx, on the Edge compared to the Cloud. This larger latency is due to the limited availability of edge node hardware, especially compared to the superior compute power of the distant Cloud. The computational cost of these applications on the Edge exceeds the communication savings from leveraging the closer edge nodes. There is a slight increase in the latency times for the Cloud-Edge benchmarks compared to the Edge Only, this is due to the time needed to transfer the model assets from the Cloud to the Edge. On the other hand less compute intensive applications such as Aeneas and FogLAMP show comparable, if not lower computational latency on the Edge when compared to the Cloud. As the execution time is lower for less compute intensive tasks, the communication savings exceeding the computational cost.

Experiment 3: Induced Stress Results & Observations:

For this experiment Gaussian workloads are simulated on the edge nodes for the Edge-Cloud pipeline for Yolo, PocketSphinx and Aeneas, and the Edge only pipeline for FogLAMP. The stress[23] package is used to induce data workload stress, this is relevant for simulating background computations. The package stress-ng[24] is also used to stress the network when capturing communication latency metrics. This experiment is used to simulate a real world use case as platforms are often noisy due to background processes and network traffic [4].

The stress and stress-ng packages use stressors to induce various levels of data and network workload stress. A stressor is a mechanism used to induce varying workload stress on a

[23]https://people.seas.harvard.edu/~apw/stress/

[24]https://kernel.ubuntu.com/git/cking/stress-ng.git/

platform or network. Minimal stress occurs when one core is stressed for the Gaussian data workload. The network stress tests involves transferring a large file of 256Mb at roughly 21740 bytes per second. For the low stress results, two CPU core stressors were invoked resulting in a greater impact on the network and service, likewise for the medium and high values. These involved stressing 3 and 4 cores respectively. Finally for the very high stress test, all four cores were stressed and the RAM module was stressed using 2 stressor processes. The compute and network benchmarks were executed multiple times for each fog application.

Each application is run five times for minimal, low, medium, high and very high levels of compute and network stress applied to the Edge. For the compute and network data loads, progressively more of the available CPU's, bandwidth and RAM modules are stressed. This is to comparatively evaluate applications running on a noisy Edge service, which is representative of a real world use case where external systems and services will be interacting with the edge nodes, and network traffic will be prevalent.

All fog applications follow the same observed trend of increasing network latency as the network bandwidth tests become more intensive, as shown in figure 9 (a). As such, it can be observed that applications that transfer larger workloads such as PocketSphinx are affected the most as the network is stressed and more bandwidth is used. It is worth noting the results are more erratic for the Odroid benchmarks, especially for applications that send smaller payloads such as FogLAMP during less intensive network workloads. This is potentially due to the network traffic of the test environment.

As highlighted in figure 9 (b), compute intensive applications such as Yolo which have a larger execution time see the greatest increase in compute latency during induced data workloads. As observed, the computational latency significantly increases as the data workloads stressors are increased. Conversely the less compute intensive applications Aenaes and FogLAMP can be observed to have comparatively less negative impact from the induced workloads. The disparity in edge node results from the previous experiments is also observed here, but the computational results become more comparable as the induced data workload stress increases.

Experiment 4: Multiple Users Results & Observations:

For this experiment, multiple benchmark runs are simultaneously executed to simulate concurrent use of the application and network. Benchmarks are run for 1, 2, 5, 10, and 50 concurrent users. The aim of this test is to evaluate the Edge's performance with multiple users interacting with the service simultaneously. This is more representative of real world use. These test cases evaluate the impact of multiple compute and network tasks on the fog system. While experiment 3 evaluated the effect of gradual and consistent workload stress, this experiment simulates interaction from concurrent users which is more erratic than artificially induced Gaussian workloads.

Regarding the communications latency, as shown in figure 10 (a), the response and time in flight metrics increase steadily with the number of concurrent users. This trend is also present for the average data transfer rate for all applications. As the number of concurrent users increases the rate at which data payloads are uploaded or downloaded decreases. This trend is outlined in figure 10 (c). The transfer rate decrease is greater for applications that transfer large asset and model files such as Yolo. These communication results adhere to the trend found in experiment 3 for a noisy platform [4].

The observed communication overheads from concurrent users was relatively small, conversely figure 10 (b) shows the computation latency and execution time observes a significant increase, particularly for computationally intensive applications. As such applications that have a large execution time for a single user, are impacted the most by concurrent benchmarks, resulting in larger computational latencies. This is particularly true for the real time factor metric, observed in figure 10 (d), which is a significantly larger when leveraging the Edge, especially compared to the Cloud. This value is observed increasing as the number of concurrent users grows, and the limited availability of the edge nodes results in an even greater impact. On the other hand, applications that prioritise frequent communication over performing compute intensive tasks such as FogLAMP, observe only a slight increase to the return trip time and compute latency metrics. The Cloud-Edge pipeline observes slightly larger values due to the time necessary to transfer the model asset.

Experiment 5: External Tools Results & Observations:

In this experiment Apache JMeter and Taurus are used to simulate end user behaviour and generate synthetic workloads [9], by sending mocked payload data to the server endpoints. This experiment focuses on the response latency and percentage of successful responses. This experiment is run for the latency critical GPS online mobile game iPokeMon. The test duration is set to 5 minutes with a 0 second ramp up period. The users are increased from 1 to 250 concurrent simulated threads. A second test was conducted for 10 concurrent users in a noisy environment, this was achieved by inducing network stress on the Edge using stress-ng, similar to experiment 3. This provides insight into the online mobile games performance during varying network conditions. This is particularly useful for online games that need to maintain a good QoE and minimise lag [19], especially for large user bases often during peak network traffic.

As highlighted in figure 11 (a) there is a large disparity between the response latency of the edge nodes, but both devices have a significantly lower average response latency than the Cloud Only solution for all simulated user threads. The edge nodes begin with roughly the same latency value for a single user, but the Raspberry Pi 3 latency remains at approximately 40 seconds until 50 concurrent users where the value spikes to roughly 120 seconds. Conversely the Odroid XU4 has a steady increase in the response latency for more than one concurrent user, until a peak latency of 130 seconds.

The same trend can be observed when inducing increasingly more intensive network bandwidth workloads during the benchmarks for 10 concurrent users, as shown in figure 11 (b). The Raspberry Pi seems relatively unaffected by the stress,

only increasing marginally for the most intensive network workload. While the Odroid XU4 tests are affected more by the network traffic bandwidth tests, the Cloud observes a significantly greater increase to latency. It is worth noting that the latency spikes observed for the edge nodes for the very high induced stress tests, was due to a long running job that failed to send a success payload to the server, likely due to the period of induced high network traffic. Overall the edge devices returned a success rate of 99% while the Cloud resulted in a lower success rate of 94%.

The standard deviation was calculated using Taurus to determine the impact and frequency of outliers on the results set. The Odroid XU4 returned an average value of 0.083078667 for 250 concurrent users, the Raspberry Pi 3 observed a slightly lower average value of 0.081034311 while the Cloud had a larger average value of 0.138751632. The larger standard deviation value of the Cloud along with the lower response success rate indicate the edge nodes perform better for a larger number of concurrent users and during periods of intense bandwidth and network traffic.

C. Discussion

Throughout these experiments a myriad of relevant metrics were captured [22], as shown in tables 2 to 5. Three categories were chosen to be the focus of these experiments, 1) Communication, 2) Compute Performance and 3) Concurrency. These metrics are important when determining if leveraging the Edge would be beneficial for a fog application [5].

From these results it is clear the Cloud provides compute benefit over the edge nodes limited availability of hardware [7]. It can then be deduced that a compute intensive application will perform better on the Cloud if the computation overheads are significantly larger than communication latency savings on the Edge. As a result the application will receive no immediate benefit from leveraging the Edge. From this it can be proposed that by spitting application functionality more generously across the Cloud and Edge, the most compute intensive tasks would be performed in the Cloud, and frequent communication as well as simple invocations would be performed on the Edge. An example of this is a fork of PocketSphinx called PocketSphinx-Python[25]. PocketSphinx is compute intensive application, but this python fork provides segmented functionality. As a result an API can be invoked to perform the less intensive task of continuous listening. By offloading less intensive tasks to the Edge, the compute latencies and real time factor may be decreased. On the other hand, fog applications that benefit from frequent communication or require larger workload assets such as a iPokeMon and Aeneas, can see great benefit from the shorter geographical distance of the Edge [2]. It can further be deduced that the communication savings for an application needs to outweigh the computation overheads for benefit to be obtained, additionally this needs to motivate the effort of potentially refactoring code to leverage a platform other than the large scale Cloud [1].

It is expected that by 2025, 80 billion edge devices will be connected. This results in an overabundance of data traffic and communications that will make the Cloud unsustainable for centralised storage and computing [23]. This is especially true for IoT and fog applications that require lower latency than what is offered by the Cloud to efficiently operate [24]. As such from these results it can be concluded that IoT application such as FogLamp can benefit from the Edge. As the use of API's to frequently retrieve and store data from local services and sensors is not compute intensive and will benefit from the close proximity of the Edge [15]. Applications such as this require less intensive computational effort but more consistent and frequent communication. Particular benefit could be seen within infrastructure, as sensor data stored locally could be retrieved or posted to the service significantly quicker than when communicating with the distant Cloud, especially during heavy network use. This is evident due to the reduced communication latency and data transfer overheads perceived in the experiments during periods of heavy network traffic and data load [2] [1].

Latency critical applications such as Aeneas would benefit from the Edge, due to the frequent communication required to supply the application with the data required to perform forced alignment. This is particularly useful for narrating and providing audio accessibility descriptions on the go. This process needs to be performed close to real time to minimise the time taken to match the audio segments with text, which is not feasible or sustainable on the distant Cloud [23]. Similarly online games such as iPokeMon would greatly benefit from leveraging the Edge. Not only are the communication overheads considerably less than with a Cloud only solution but when leveraging the Edge it was observed that less failed responses ocurred for intensive periods of network traffic. This is important for perserving an online games QoE and QoS [9].

Deep learning applications such as Yolo could potentially see benefit if the compute tasks were split more generously between the Cloud and the Edge, with the demanding tasks such as model training and classification being performed on the Cloud, and the test data frequently supplied to the Edge. An important consideration for these results, especially for deep learning systems and applications, is the trade-offs between training time, training cost, and inference time [22]. It can be deduced that for computationally expensive training, that this would not be particularly beneficial if performed on the Edge. As such these experiments assume that the training has occurred on the Cloud beforehand. The Cloud-Edge pipeline in particular makes use of this assumption, as the weights and acoustic models are transferred from the Cloud platform where the training is assumed to have been completed.

VII. Related Work

In Mobile Cloud Computing (MCC) applications offload computationally intensive tasks and extend their edge device functionality using the Cloud, this differs from Edge computing which makes use of edge nodes to offload compute resources which are geographically closer from the user device than the Cloud [10]. The paper lacks a wide range of use cases and workloads. 'Benchmarker' is an example of a benchmarking tool used to execute tasks on the Cloud and

[25]https://pypi.org/project/pocketsphinx/

TABLE VII: Benchmark Tool Taxonomy.

Name	Type	Platform	Metrics
PerfKitBenchmark	Benchmark Suite	Clouds	Cloud offerings and compute resources
iPerf	Network Performance Tool	Cross-platform	I/O and network (TCP/UDP)
wakeword-benchmark	Benchmark Framework	Wake-word detection engines	Accuracy and detection rate
EdgeBench	Benchmark Framework	IoT Edge platforms	Latency, CPU and memory utilisation
LINPACK	Benchmark Tool	Supercomputers	Floating point comparisons
stt-benchmark	Benchmark Framework	Speech to text engines	Word accuracy rate, speech conversion rate, memory usage
Tailbench	Benchmark Suite	Latency critical applications	Load-latency and throughput centric metrics.
hyperfine	Benchmark Tool	Cross-platform	Application performance and statistical outlier detection
nench	Benchmark Script	Cross-platform	CPU, I/O and network (ioping tests)
Unixbench	Benchmark Suite	Unix-like platforms	CPU, parallel processing and concurrency centric metrics.
DCBench	Benchmark Suite	Data center workloads	Bespoke metrics: segmentation, clustering and classification
Sysbench	Benchmark Tool	Cross-platform	OS parameters under data intensive loads
Phoronix-test-suite	Benchmark Suite	Cross-platform	Platform performance testing and continious integration
bench-sh-2	Benchmark Script	Unix systems	Network, CPU and I/O
Apache JMeter	Testing Tool	Cross-platform	Load and performance testing
Taurus	Automated Test Framework	Cross-platform	Platform performance and concurrency testing
DAWNBench	Benchmark Suite	Deep learning applications	End to end latency and training inference
Benchmarker	Benchmark Tool	Bespoke projects	Bespoke metrics for program statistical evaluation

user device with little set-up time. A taxonomy of relevant benchmark suites and frameworks is outlined in table 7. The approach this research proposes is to develop a benchmarking method for the evaluation of fog systems across the Cloud and Edge, this aims to offer insights that can be used to maintain the QoE and QoS for online games [9], deep learning and IoT applications leveraging the Edge. DeFog integrates many of the features outlined in the taxonomy, as DeFog is an open source automated cross platform command line tool that evaluates containerised fog applications and platform services on the Cloud and Edge.

Typically benchmarking applications on the Cloud is done by executing a set of standardised tests using benchmark applications. Applications such as PerfKitBenchmarker, iPerf or Linpack obtain metrics from various Cloud resources to evaluate relative performance [4]. This implementation focuses on benchmarking cloud virtual machines using a set of weights that are relevant to resources such as: local communication, memory and computation. This approach may not be appropriate on the Edge, as cloud-based solutions require a lot of time for benchmarking. The Edge is a more dynamic and versatile system, and benchmarking results need to be generated quicker. Additionally, it is not clear whether the metrics obtained on the Cloud would be at all relevant to the Edge. The research in this paper proposes containerised benchmarking to capture beneficial metrics on the Edge.

Peering Edge Networks in particular have the potential to benefit the users of applications such as massively multiplayer online games by reducing latency and screen jitter due to the close proximity of the edge [20]. Pushing application logic to Peer to Peer (P2P) servers typically has resulted in unreliable connections and peer trust issues. Google's Edge network is an alternate technique that offers a solution for availability, saturation and security. This technique allows developers to exploit the Edge to offload application logic to a low latency scalable system closer to the client. This approach is limited to a single desktop multiplayer game use case which is typically played using a wired Ethernet connection, and not "on the go". The important metrics for desktop games differs greatly than for mobile games, as mobile games typically require wireless connections to transfer data as the user physically moves location. The research proposed in this article benchmarks a more Edge relevant mobile game that along with a selection of deep learning and IoT applications giving insight into a wider range of applications and workloads on the Edge.

Edge nodes such as routers and switches reside closer to the user than the Cloud, Edge NOde Resource Management [7] proposes a framework for managing such edge nodes through leveraging computing at the Edge, closer to edge devices [7]. This is to reduce latencies and data traffic to the Cloud. JMeter is used to load test the open source mobile online game iPokemon. The research proposed in this report will extend this use case by integrating third party systems such as JMeter and Taurus within the proposed method to autonomously generate a plethora of metrics communication latency autonomously to evaluate network performance of the Edge.

Motivations to leverage edge computing include overcoming the resource limitations of front-end devices, sustainable energy consumption and providing a platform for low latency computing [5]. Yet several challenges exist with edge-based solutions, such as: partitioning processes, offloading tasks and security issues when using public edge nodes. The aim of this proposed method is to evaluate such motivations by evaluating a catalog of insightful metrics.

Energy consumption is an important motivator as the increase of power-hungry applications is not matched by parallel improvements in users device's battery life. By offloading the most demanding tasks to the mobile edge computing network, energy performance benefits can be achieved [17]. MVR, the architecture presented through fine grain offloading of code to the Edge makes use of virtual resources to reduce energy consumption of edge devices. But savings gained from code offload need to exceed the cost of additional communications [25]. Netperf TCP streaming benchmark is used to gather metrics for obtaining the energy consumption of wireless communication. The extensibility of the proposed method allows for future benchmarks to capture a plethora of new metrics such as energy consumption.

For benchmarks to be representative of real world application particularly for online games, typically player behaviour

must be simulated, this is often referred to as a 'synthetic workload'. A developer's understanding of the application's performance for varying user behaviour is important to the QoE, as imperfections such as screen jitter can lead to noticeable delays and inconsistencies in a player's game state [9]. The proposed method uses synthetic workloads to capture metrics relevant to an online game's QoE.

Another system that maximises the benefit and QoS from metrics such as energy savings, through remote execution is MAUI. MAUI is a system that allows for fine-grained remote code offload to an external infrastructure. This leverages the benefits of a managed code environment for code offload determined at run time [26]. MAUI's approach explores a desktop game use case, this simulates data offload using a selection of Microbenchmarks and Macrobenchmarks. Deterministic code offload may be beneficial to improving the QoE of online games, but MAUI is focused on energy consumption attributes while the proposed research captures a catalogue of metrics for a larger variety of fog applications: mobile games, deep learning applications and IoT applications.

CloudRank-D is the benchmarking suite proposed to evaluate and rank Cloud computing systems that run big data applications by proposing two metrics: data processed per second and data processed per Joule as complementary metrics [3]. These metrics may not be relevant on the Edge, this research analyses a range of different workloads and captures various metrics pertaining to performance, communication and concurrent user impact.

IBM Shadow Puppets proposes a semantic cache, by blending features of the extreme ends of the design space. This results in inference latency reduction for object classification [11]. Moving IoT application sensor data to the Cloud is not economically viable as it increases the cost of insights and diminished returns time to insight latency. This proposed method evaluates an IoT application leveraging the Edge. The paper proposes benchmark method for evaluating the performance of deep learning object detectors Iris and YOLO [27]. The focus of this paper regarding the compute performance for different convolution kernels [14] [28] may not be applicable on the Edge, due to the limited availability of hardware required for intensive machine learning computation.

Edge bench is the benchmarking tool proposed the evaluate performance applications running on IoT Edge services [2]. The paper makes use of pipelines to generate metrics on IoT platforms Amazon Greengrass and Azure IoT Edge. The research proposed in this paper performs fog application benchmarks using simulated edge nodes.

An analysis of the performance and scalability of a hybrid Edge-Cloud system which supports latency-sensitive applications is proposed in the paper, this evaluates design parameters such as latency and goodput [1]. An Artificial reality use case example is considered using latency under load stress. A similar strategy is adapted in this research to induce data workloads across the network and platform during benchmarking.

VIII. Conclusion

This paper has presented DeFog, a benchmarking method for containerised applications deployed across the Cloud and the Edge. Five applications workloads have been evaluated: Yolo, PocketSphinx, Aeneas, FogLAMP and iPokeMon, deployed across three distinct pipelines: Cloud Only, Edge Only and the combined Cloud-Edge. The performance of these applications has been evaluated with a series of experiments that capture a catalog of relevant metrics [22] for a standard execution, concurrent use and induced workloads on Odroid XU4 and Raspberry Pi 3 boards used to simulate edge nodes.

It is crucial that the time for running benchmarks on the Edge is minimised [4]. This is achieved by using Docker to automate builds and run detached containers. The isolated containers ensure each fog application can be run within identical environments with the same dependencies resolving to more consistent benchmarks. The deployment and benchmarks are automated, ensuring minimal human interaction and same step execution. An important consideration is the relative ease of deploying the applications, workloads, packages and dependencies across the Cloud and Edge using DockerFiles. This is in a stark contrast to the likes of IoT Edge platforms such as Amazon Greengrass that require external dependencies compiled and added as a zip file for deployment [2].

The experimental results show that several applications can benefit from leveraging the Edge, due to the reduced cost of communication overheads. But this benefit is reduced for applications that require intensive computations. It is for this reason it can be concluded that for a fog application to benefit from leveraging the Edge, the application needs to be componentised by generously splitting the compute intensive components across the Cloud and Edge, to reduce the complex computations performed on the Edge. Regarding iPokeMon, a 99% successful response rate was observed when leveraging the Edge, while the Cloud benchmarks resulted in a lower 94% success rate. It is crucial for online games such as this to preserve consistency and reduce latencies to maintain a high QoS and QoE [9], as observed when leveraging the Edge.

It can also be concluded that while there is a slight disparity between the compute results generated on both edge nodes, the same trend can be observed for all fog applications leveraging the Edge. Concurrency is also an issue on the Edge, as the limited hardware becomes strained during intensive data workloads [5]. On the other hand, communication latency is impacted less from network traffic, especially when compared to the Cloud. It can be concluded that benefit can be gained by applications that perform less intensive compute tasks and prioritise frequent communication with the Edge.

A. System & Research Limitations

A few notable limitations exist with the benchmarking method. Computation offloading is not optimally utilised, as the implementation could benefit from splitting functionality more generously between the Cloud and Edge. It was observed that for a large number of concurrent users, the edge nodes were becoming overloaded due to the frequent compute intensive workloads [5]. Ideally any compute intensive tasks would be performed on the Cloud while less intensive actions such as continuous listening would be performed on the Edge. Security and privacy is often determined to be a benefit within

fog computing, but addressing the security operations required on the edge nodes to extend or offload functionality is an important consideration [23]. Secure Shell and Secure Copy Protocol are used in place to ensure secure connections are maintained. This may become an issue in the future as suitable storage capabilities will need to be available on the Edge.

Several Cloud instances could have been set up using different deployment regions for further evaluation. In addition to this another limitation was the local network between the user device and the edge nodes. This network was often inconsistent and relatively slow. The Odroid XU4 and Raspberry Pi 3 used to simulate the limited availability of hardware of the Edge may not be entirely reflective of the computational power available on a typical edge node. Other end devices could potentially have been evaluated to provide a greater range of evaluative metrics. While the cost of running the application benchmark is captured for the Cloud, the Edge cost is only estimated. This cost attribute could be greatly expanded upon in the future to generate actionable insight into the comparative performance cost of the Edge, compared to the Cloud.

IoT application FogLAMP was only integrated for the Cloud and Edge Only pipelines. This implementation mocks sensor connections resulting in limited API functionality. Potential exists to deploy sensor services or PostgresSQL databases to simulate real world use. A plethora of games and applications were researched and partially integrated within DeFog before determining their implementation to be impractical or irrelevant to Fog computing. Open source online games such as Xonotic, OpenTTF, Ancient Beast, AR-Madness, BZFlag, minetest and freeciv-web were determined to be not feasible for the Edge due to their complex code base and lack of documentation. Microservice application TeaStore, deep learning tool Iris and speech recognition tools Merlin, Sphinx and mozilla-DeepLearning were also reviewed, these applications informed the use YOLOv3 and PocketSphinx.

B. DeFog: Future Work

As DeFog is an open source project, potential exists to extend functionality by integrating new fog applications, endpoints, platforms and metrics. The intuitiveness of Bash shell scripting with python is beneficial as this provides fast performance, interoperability, as well as a low mean time to refactor and integrate robust security features [5]. DeFog is only the beginning for Fog system benchmarking, as there is scope to extend the catalog metrics captured and integrate new systems to generate and calculate attributes such as energy consumption [25]. As research pertaining to the componentising and splitting of application functionality across the Cloud and Edge progresses, DeFog will become even more appropriate for gathering insights pertaining to the benchmarking of fog application compute resources on the Edge.

REFERENCES

[1] Maheshwari, Sumit & Raychaudhuri, Dipankar & Seskar, Ivan & Bronzino, Francesco. (2018). Scalability and Performance Evaluation of Edge Cloud Systems for Latency Constrained Applications, 2018

[2] Das, Anirban & Patterson, Stacy & Wittie, Mike.(2018). EdgeBench: Benchmarking Edge Computing Platforms.

[3] Luo, Chunjie & Zhan, Jianfeng & Jia, Zhen & Wang,Lei & Lu, Gang & Zhang, Lixin & Xu, Cheng-Zhong & Sun,Ninghui. (2012). CloudRank-D: Benchmarking and rankingcloud computing systems for data processing applications. Frontiers of Computer Science. 6, 2012.

[4] B. Varghese, O. Akgun, I. Miguel, L. Thai and A.Barker, Cloud Benchmarking For Maximising Performance of Scientific Applications, IEEE Transactions on Cloud Computing, pp. 1-1, 2016.

[5] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick and D. Nikolopoulos, Challenges and Opportunities in Edge Computing, 2016 IEEE International Conference on SmartCloud, 2016.

[6] C. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. Bailis, K. Olukotun, C. R, and M. Zaharia. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. NIPS ML Systems Workshop, 2017.

[7] N. Wang, B. Varghese, M. Matthaiou and D.Nikolopoulos, ENORM: A Framework For Edge NOde Resource Management, IEEE Transactions on Services Computing, pp. 1-1, 2017.

[8] Jacobs, Marco C., and Mark A. Livingston. Managing latency in complex augmented reality systems. Proceedings of the 1997 symposium on Interactive 3D graphics. ACM, 1997.

[9] T. Triebel, M. Lehn, R. Rehner, B. Guthier, S. Kopfand W. Effelsberg, Generation of synthetic workloads for multiplayer online gaming benchmarks, 2012 11th Annual Workshop on Network and Systems Support for Games(NetGames), 2012.

[10] N. Naqvi, T. Vansteenkiste-Muylle and Y. Berbers,Benchmarking leading-edge mobile devices for data-intensive distributed mobile cloud applications, IEEE Symposiumon Computers and Communication, 2015.

[11] Srikumar Venugopal, Michele Gazzetti, Yiannis Gkoufas, and Kostas Katrinis, Shadow puppets: Cloud-level accu-rate AI inference at the speed and economy of edge, USENIX workshop on hot topics in edge computing (hotedge 18), 2018.

[12] Armbrust, Michael, et al. A view of cloud computing. Communications of the ACM 53.4 (2010): 50-58.

[13] M. Satyanarayanan, The emergence of edge computing, Computer, vol. 50, no. 1, pp. 3039, 2017.

[14] J. Dai, Y. Li, K. He, and J. Sun. R-FCN: Object Detection via Region based Fully Convolutional Networks. In NIPS, 2016.

[15] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, Fog Computing and Its Role in the Internet of Things, in Proceedings of the Workshop on Mobile Cloud Computing, 2012, pp. 1316.

[16] J. Redmon and A. Farhadi, YOLO9000: Better, Faster, Stronger, 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[17] X. Wei, S. Wang, A. Zhou, J. Xu, S. Su, S. Kumar andF. Yang, MVR: An Architecture for Computation Offloading in Mobile Edge Computing, 2017 IEEE International Conference on Edge Computing (EDGE), 2017.

[18] H. Kasture and D. Sanchez. Tailbench: A benchmark suite and evaluation methodology for latency-critical applications. In IEEE International Symposium on Workload Characterization (IISWC). IEEE, 2016.

[19] Zhang, Wuyang, et al. Towards efficient edge cloud augmentation for virtual reality MMOGs. Proceedings of the Second ACM/IEEE Symposium on Edge Computing. ACM, 2017.

[20] J. Plumb and R. Stutsman, Exploiting Googles Edge Network for Massively Multiplayer Online Games, 2018, IEEE 2nd International Conference on Fog and Edge Computing (ICFEC), 2018.

[21] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, Edge computing: Vision and challenges, IEEE Internet Things J.,vol. 3, no. 5, pp. 637646, 2016.

[22] Z. Li, L. OBrien, H. Zhang and R. Cai, On a Catalogue of Metrics for Evaluating Commercial Cloud Services, 13th International Conference on Grid Computing, 2012, pp.164- 173.

[23] O. Rana, M. Shaikh, M. Ali, A. Anjum and L. Bittencourt, "Vertical Workflows: Service Orchestration across Cloud & Edge Resources," 2018 IEEE 6th International Conference on Future IoT and Cloud, 2018.

[24] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V.Young, Mobile edge computing, A key technology towards 5G,ETSI white paper, vol. 11, no. 11, pp. 116, 2015.

[25] Antti P. Miettinen , Jukka K. Nurminen, Energy efficiency of mobile clients in cloud computing, Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, p.4-4, 2010, Boston, MA.

[26] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S.Saroiu, R. Chandra and P. Bahl, MAUI, Proceedings of the8th international conference on Mobile systems, applications,and services - MobiSys 10, 2010.

[27] E. Severo et al., A Benchmark for Iris Location and aDeep Learning Detector Evaluation, 2018 International Joint Conference on Neural Networks (IJCNN), 2018.

[28] A. Ignatov, R. Timofte, P. Szczepaniak, W. Chou, K. Wang, M. Wu, T. Hartley, and L. Van Gool, AI Benchmark: Running Deep Neural Networks on Android Smartphones, (2018).