DeFog: Fog Computing Benchmarks

Jonathan McChesney
IBM, UK
jonathan.mcchesney@ibm.com

Nan Wang
Queen's University Belfast, UK
nwang03@qub.ac.uk

Ashish Tanwer
Cisco Systems, USA
astanwer@cisco.com

Eyal de Lara
University of Toronto, Canada
delara@cs.toronto.edu

Blesson Varghese
Queen's University Belfast, UK
b.varghese@qub.ac.uk

ABSTRACT

There are currently no benchmarks that can directly compare the performance of an application across the cloud-only, edge-only and cloud-edge (Fog) deployment platforms to obtain any insight on potential performance improvement. This paper proposes DeFog, a first Fog benchmarking suite to: (i) alleviate the burden of Fog benchmarking by using a standard methodology, and (ii) facilitate the understanding of the target platform by collecting a catalogue of relevant metrics for a set of benchmarks. The current portfolio of DeFog benchmarks comprises six relevant applications conducive to using the edge. Experimental studies are carried out on multiple target platforms to demonstrate the use of DeFog for collecting metrics related to application latencies (communication and computation), for understanding the impact of stress and concurrent users on application latencies, and for understanding the performance of deploying different combination of services of an application across the cloud and edge. DeFog is available for public download (https://github.com/qub-blesson/DeFog).

CCS CONCEPTS

• **Computing methodologies** → **Distributed computing methodologies**; • **Computer systems organization** → *Embedded and cyber-physical systems*; • **Software and its engineering** → *Software notations and tools.*

KEYWORDS

fog computing, benchmarking, edge computing, containers

ACM Reference Format:
Jonathan McChesney, Nan Wang, Ashish Tanwer, Eyal de Lara, and Blesson Varghese. 2019. DeFog: Fog Computing Benchmarks. In *SEC '19: ACM/IEEE Symposium on Edge Computing, November 7–9, 2019, Arlington, VA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3318216.3363299

Mr Jonathan McChesney contributed to the research as a Master's student at Queen's University Belfast, UK, under the supervision of Dr Blesson Varghese.

SEC '19, November 7–9, 2019, Arlington, VA, USA

ACM ISBN 978-1-4503-6733-2/19/11…$15.00
https://doi.org/10.1145/3318216.3363299

1 INTRODUCTION

The premise of the Fog computing is to bring appropriate services of an application from the cloud to the edge of the network [3, 27]. Since the edge is closer to end user-devices, running services on edge resources will reduce the communication latencies of applications by enabling more data to be processed near an end user-device before it is sent to (or processed on) the cloud [22, 24, 30]. Consequently, the overall Quality-of-Service (QoS) and Quality-of-Experience (QoE) of an application can be improved [4, 8].

Given that Fog computing is in its nascent stages of development, there are a number of challenges that need to be addressed. One such challenge is understanding the relative performance of Fog applications by comparing target hardware platforms from different vendors due to diversity of hardware architectures and the impact on performance when system software level changes or new networking protocols are introduced. Fog benchmarking solutions are required to address this.

Benchmarking is a commonly used technique for evaluating the relative performance benefits of different target computing models (or platforms) and the applications that run on them [16, 26]. Benchmarks capture a variety of workloads that are likely to be executed using a computing model. These workloads are systematically and repeatedly executed so as to obtain a large catalogue of relevant metrics related to performance, which is subsequently analysed for obtaining concrete answers to questions posed by hardware vendors or system software developers. There are dedicated benchmarks that have been developed for supercomputers, namely LINPACK [7] and NAS Parallel Benchmarks [1], or for the cloud, namely CloudRank-D [17] and DCBench [13]. However, we note that there are no such benchmarking solutions available for Fog computing. Therefore, the focus of the research presented in this paper is to report the development of a first Fog benchmarking suite, referred to as DeFog (the rationale for the name is to 'demystify cloud-edge interactions of a Fog system').

The DeFog suite is underpinned by a six step benchmarking methodology that operates in three deployment modes, namely a cloud-only mode, edge-only mode, and cloud-edge (Fog) mode. We anticipate that DeFog can be used to understand at least the following three fundamental questions:

Q1: How can the relative performance of using a Fog platform over the cloud-only platform be understood and quantified?

Q2: If there are multiple services of an application that can be moved to the edge, then how can performance of using the Fog be understood?

Q3: If there are multiple competing resources available at the edge suited for a specific service, then which resource should be selected for maximising the overall performance gain?

Benchmarking across different deployment modes will address ***Q1***. DeFog is as hardware agnostic as possible (to the extend current container technology supports) in building and deploying the cloud and edge services of the Fog application. The methodology captures a catalogue of performance metrics that are related to the target platform (cloud and edge resources) and applications running on the platform. It would also be possible to obtain insight into the best distribution of services for a given class of application across the cloud and the edge, thereby addressing ***Q2***.

Six applications that are Fog relevant workloads, namely a deep learning-based object recognition, a text-to-speech converter, a text-to-audio forced alignment, an online mobile game, an Internet-of-Things (IoT) application, and real-time face detection from video streams, are considered. The applications are latency critical, stream-based and/or bandwidth intensive, thereby making them ideal candidates for use in DeFog. The experimental results presented highlight the relative performance of the benchmarks across the deployment modes on different target platforms. This demonstrates the feasibility of DeFog for addressing ***Q3*** considered above. The DeFog software is publicly available[1].

The contributions of this paper are as follows: (i) The development of a platform agnostic Fog benchmarking methodology, DeFog, that operates in three deployment modes. (ii) The evaluation of six application benchmarks that are relevant to Fog computing and making them available in a repository. (iii) The identification and collection of a catalogue of metrics that capture the properties of the target platform and the application running on it. (iv) An experimental evaluation of DeFog across cloud resources and multiple single board computers that are used as edge resources.

The remainder of this paper is organised as follows. Section 2 provides an overview of the benchmarking methodology and the deployment modes of DeFog. Section 3 presents DeFog's current portfolio of six benchmarks that are candidate Fog applications and the catalogue of metrics that are collected by DeFog. Section 4 presents the results obtained from an experimental study. Section 5 considers the related work. Section 6 concludes this paper by presenting future work.

2 DEFOG BENCHMARKING

This section firstly presents a few observations that have led to the design and development of DeFog, followed by the benchmarking methodology that is incorporated in DeFog, and finally the deployment modes of DeFog. The first version of DeFog is available for download from (https://github.com/qub-blesson/DeFog).

2.1 Motivation

Fog benchmarking as a research area is still in its early stages of development. The dependencies between cloud-edge services of an application are not fully known given that there are only a few open source Fog applications available. The following five general observations regarding Fog benchmarking have been considered while developing DeFog:

(i) *Fog benchmarking is complex as dependencies between cloud and edge services of a Fog application need to be considered*: In cloud benchmarking, the dependencies of the application are mostly in the cloud. However, the dependencies between cloud and edge services of a Fog application will need to be considered for Fog benchmarking. This makes Fog benchmarking more complex than cloud benchmarking.

(ii) *Fog benchmarks are not readily available*: There is a limited understanding of the real workloads that can benefit the most from using Fog computing. Consequently, there are no open source Fog benchmarks readily available. While the portfolio of benchmarks for traditional computing platforms, such as high-performance clusters or even the cloud are diverse and comprehensive, Fog benchmarks are not yet developed. DeFog attempts to create an early repository of six benchmark applications that can be expanded upon by the community as our understanding of Fog applications evolves and as applications become available.

(iii) *Fog benchmarking should generate rapid results*: Ideal benchmarking should generate results quickly and in the context of the Fog it is essential given that the edge is a transient environment. Resources located on the edge (routers or gateways made accessible for general purpose computing) are anticipated to have intermittent profiles when compared to dedicated resources in a data center. DeFog is therefore designed to execute in a lightweight manner and under one minute.

(iv) *Metrics captured during Fog benchmarking must be generalised to a wide variety of workloads*: Many benchmarking solutions are workload specific and therefore generate workload specific metrics on a target platform. Hence, they cannot provide insight into the suitability of the target platform for a different class of workloads. The aim of DeFog is to evolve over time by adding more workloads so that a wide range of metrics can be captured for diverse Fog platforms and application benchmarks.

(v) *Benchmarking needs to be consistent*: To ensure that benchmarking is consistent each execution of the benchmark must be on the same application build version and package dependencies. This is ensured in DeFog so that cloud-edge services are consistently benchmarked. In addition, to minimise the impact of any noise within the Fog platform (due to temporary network congestion or spikes in workloads), the benchmark needs to be executed multiple times for a range of input data using identical containers to obtain an averaged values of metrics.

2.2 Benchmarking Methodology

The aim of DeFog is to automate the deployment of benchmark applications on the target platform, the transfer of assets required for these applications to run on the target platform, the generation of metrics relevant to the target platform and benchmark, and finally gather results. The proposed benchmarking methodology used by DeFog accounts for these and is a sequence of six steps, which is as follows: (1) Build and run the benchmark application container image for the target platforms, (2) transfer the required asset to the cloud resource on the target platform, (3) transfer the required asset to the edge resource on the target platforms, (4) execute the benchmark application, (5) gather the values for the catalogue of metrics, and (6) return the metrics to the user. The third step is

[1] https://github.com/qub-blesson/DeFog

required for the edge-only or cloud-edge (Fog) deployment modes. The second to fifth step is repeated multiple times for each application for consistency. The metrics are measured by an observing system. The methodology in detail is as follows:

Step 1 - Build and run a container: In this first step, a container image is built and tagged for a target platform (cloud and/or edge) to ensure that a consistent execution environment is available for all future runs of the benchmark. This research employs Docker containers and it is run in the detached mode to allow for concurrent benchmarking tasks to be executed on other application containers.

Step 2 - Transfer assets to the cloud: An asset that is transferred to the cloud is the input data required by the application's service that runs on the cloud. Each benchmark application has its corresponding asset and a single benchmark application may have multiple assets. For example, there are currently six benchmark applications in DeFog and each application has its own assets. In addition, each application may have multiple assets corresponding to similar or different sizes or types of input. This ensures that DeFog can be used for varying inputs of the application to identify performance gain under different operating conditions for the application. Some of the communication related metrics discussed in a subsequent section are also measured during this step.

Step 3 - Transfer assets to the edge: The asset transferred in this case is the input data required by the application's service that will be hosted on the edge resource. Currently, the data required by the application benchmark has to be manually partitioned and provided as assets. Automation within this step is desirable, but is not within the scope of this article. Some of the communication related metrics discussed in a subsequent section are measured during this step.

Step 4 - Execute the benchmark application: Given the running container (from Step 1) and the assets (Step 2 and Step 3) for each application the number of times the benchmark needs to be executed is determined. If there are N assets for an application, then the benchmark is executed N times. The benchmark applications are further considered in Section 3.1. Additional benchmarking tools, namely UnixBench[2] and Sysbench[3] are included within DeFog. They provide a large number of target platform related metrics, including CPU performance, concurrency and I/O read write.

Step 5 - Gather the values for a catalogue of identified metrics: A combination of communication and computation related metrics obtained from DeFog. They can be categorised as: (i) target platform metrics, which are attributes that capture the characteristics of the specific target platform under consideration, (ii) Fog application performance metrics, which are attributes that provide insight into the performance of the application on the target platform, and (iii) metrics obtained from external tools, which are attributes obtained by using third party tools. The metrics will be discussed further in Section 3.2.

Step 6 - Provide the metrics to the user: The target audience that can benefit from DeFog, include (i) vendors of new edge hardware who want to demonstrate the benefit of using the Fog via benchmarks, (ii) Internet Service Providers (ISPs) who want to deploy micro data centres at the edge of the network and want to tabulate performance of Fog applications for their customers, (iii) system software administrators who want to investigate the impact on Fog applications when a system level change, such as update to operating system or libraries, is required on the edge resource, and (iv) network administrators who want to quantify the performance of Fog applications when a new networking protocol is introduced. When the benchmarks have completed execution the metrics are provided to the observing system in the form of both comma separated and verbose text files.

[2]https://github.com/kdlucas/byte-unixbench/tree/master/UnixBench
[3]https://github.com/akopytov/sysbench

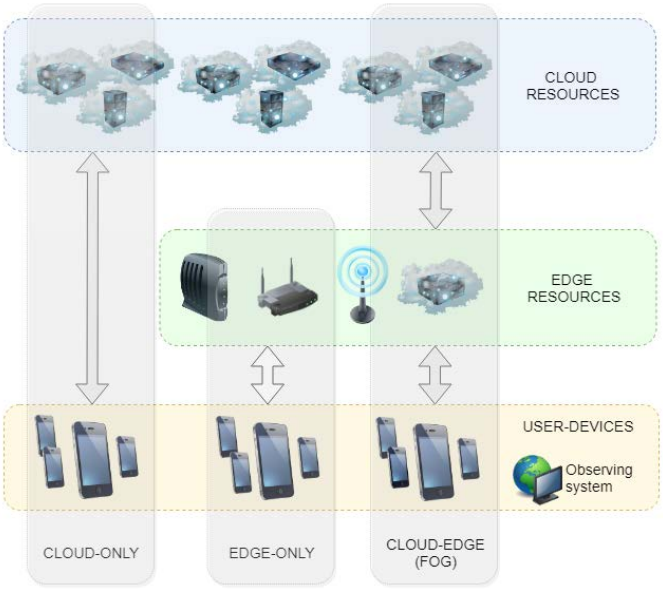


Figure 1: DeFog deployment modes, namely cloud-only, edge-only and cloud-edge (Fog).

2.3 Deployment Modes

The DeFog benchmarking suite works across three distinct deployment modes, namely a cloud-only, edge-only, and cloud-edge (Fog), as shown in Figure 1 and considered below. The underlying principle is that the relative performance of the benchmark applications should be compared on different target platforms to quantify any benefit of leveraging the edge for a cloud application.

The resources available for the target platform as shown in Figure 1 are from: (i) the cloud resource layer in which a large amount of computational and storage resources are available, (ii) the edge resource layer, which comprises either dedicated micro clouds or traffic routing nodes, such as gateways, routers and base stations, that are augmented with computing and/or storage capabilities, and (iii) the user devices layer, which comprises devices, such as smartphones, wearables, and sensors. A computer system is utilised to observe the benchmarking process and collect the metrics.

(i) Cloud-only deployment: The cloud-only deployment is typical of conventional cloud applications in which all requests originating from an end user-device are serviced by a cloud resource. Figure 2 shows the two tier cloud-only deployment mode comprising the cloud resources and user devices. In DeFog, the application is built and deployed on the cloud resource using Docker containers.

The application is then executed in the container and the benchmark metrics are generated. The method adopted is based on a container-based cloud benchmarking approach previously reported [28]. The user device uses the secure shell to interact with the cloud container during the benchmarking process.

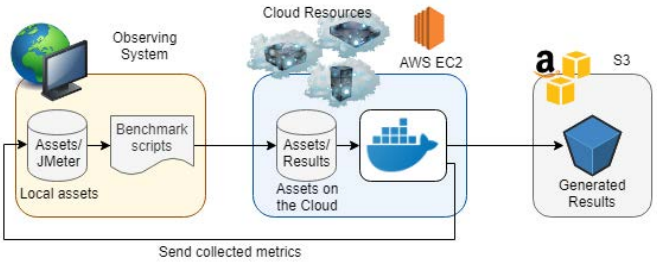


Figure 2: The DeFog cloud-only deployment mode.

A DeFog user specifies which applications in the repository need to be benchmarked and the accompanying asset is transferred to the cloud (in this research we used Amazon Web Services (AWS) Elastic Compute Cloud (EC2)[4]). The output data from the application is uploaded to an AWS Simple Secure Storage (S3)[5] bucket. The output is also transferred to the observing system along with the metrics generated during the benchmarking process.

(ii) Edge-only deployment: The edge-only deployment mode assumes that all services of an application can be entirely run on an edge resource as shown in Figure 3. This is practical if it is assumed that dedicated micro clouds or modular data centres are located at the edge of the network and the application provider replicates the application on multiple geographic locations closer to the end user. Similar to the cloud-only deployment mode, the application is deployed on the the edge resource using Docker containers. The application is then run within the container and metrics are generated. In this research resource constrained single board computers, such as Odroid XU4[6] and Raspberry Pi 3[7] are used as edge resources. The outputs and the metrics are stored in an S3 bucket as well as sent to the observing system.

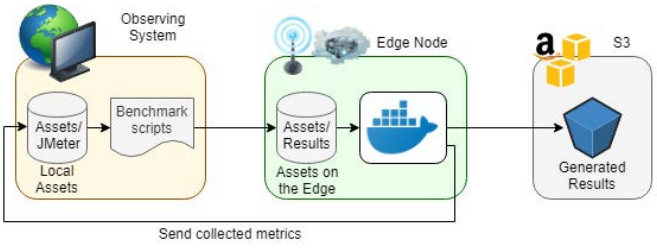


Figure 3: The DeFog edge-only deployment mode.

(iii) Cloud-edge (Fog) deployment: In the Fog deployment mode, services of an application may be distributed across the cloud and edge as shown in Figure 4. Communication latency or bandwidth sensitive services of the application are offloaded to the edge so that the overall QoS of the application can be maximised.

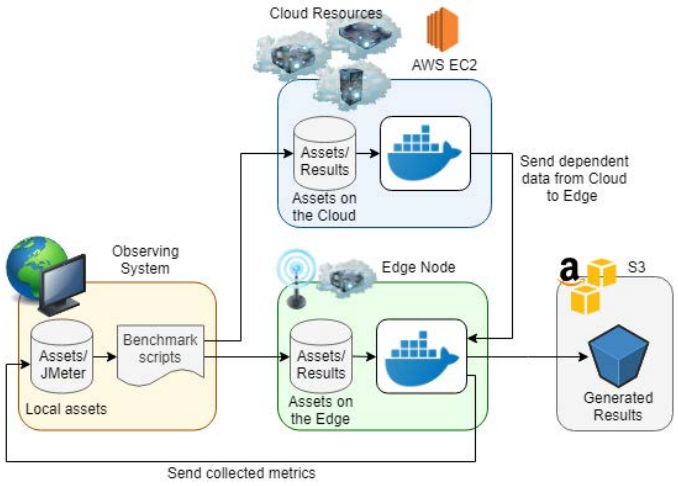


Figure 4: The DeFog cloud-edge (Fog) deployment mode.

Consider for example a deep learning application for object detection. This application will require the training of a model, which is a computationally intensive task, and cannot be easily carried out on the edge. If large micro cloud like resources are assumed at the edge, then it may be possible. However this will not be the case if there are only smaller form factor and resource constrained embedded devices available on the edge. Therefore, the training service of the application is ideally suited for the cloud. The detection service could be offloaded to the edge. The assets required by this service are a trained model and associated weights which will need to be offloaded from the cloud to the edge.

3 BENCHMARK APPLICATIONS AND METRICS COLLECTED BY DEFOG

In this section, six applications that are currently used as benchmarks in DeFog. These applications are chosen based on our current understanding of the different axes that need to be tested to characterise the benefit of using a cloud-edge deployment model. The axes tested include communication latency and computation latency with respect to concurrent users, Fog system stress and network stress, data transfer (uplink and downlink) rates with respect to concurrent users and Fog system and network stress, and improvement in responsiveness by using a Fog system against alternate deployments, such as cloud-only.

Table 1) summarises the applications and shows the coverage of the benchmarks currently available. Latency critical, bandwidth intensive, location aware and computational intensive workloads are represented. In addition, applications with single and multiple users (in many cases multiple users are simulated) are considered. Most applications only use a single edge or cloud resource. However, one application which uses multiple edge and/or cloud destinations is presented. Applications that can offload multiple services to the edge are considered. In addition, assets required to be transferred

[4]https://aws.amazon.com/ec2/
[5]https://aws.amazon.com/s3/
[6]https://magazine.odroid.com/odroid-xu4
[7]https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

Table 1: Fog application benchmarks used in DeFog; Type - Latency Critical (LC), Bandwidth Intensive (BI), Location Aware (LA), Computational Intensive (CI); Single - S, Multiple - M, Multiple, simulated - M(S), Multiple possible, not presented M(N)

Application	Description	Type	End device	Destination		Edge services	Asset transferred from cloud to edge for offload
				Edge	Cloud		
YOLOv3	Object classification using deep learning	BI, CI	S, M(S)	S	S	S	Trained model and weights
PocketSphinx	Speech-to-text conversion	BI, CI	S, M(S)	S	S	S	Trained acoustic model
Aeneas	Text-audio forced alignment	BI	S, M(S)	S	S	S	Text segment
iPokeMon	Geo-location based online mobile game	LC, LA	S, M(S)	S	S	S	Location specific data
FogLAMP	IoT edge gateway application	LC	M	M	M	M(N)	NA
RealFD	Real-time face detection on video streams	LC, BI, CI	S	S	S	M	Depends on edge services

from the cloud to the edge for successfully executing an offloaded service is presented. Finally, the metrics collected using DeFog are highlighted.

3.1 Benchmark Applications

The original version of the applications had to be modified to suit the deployment modes reported in the previous section. The modified version of the applications used in DeFog is available in the project repository[8]. Additional workloads can be added to this portfolio to enhance DeFog.

Application 1 - Deep learning based object classification using YOLOv3: The You Only Look Once (YOLO)[9] is a real-time deep learning based object detection system that uses a single neural network for the entire image [20]. The image is decomposed into different regions and the bounding boxes and probabilities for each region are estimated. These bounding boxes are weighted by the predicted probabilities. The YOLOv3 [21] is a faster system compared to competing systems that implement Region-based Convolutional Neural Networks (R-CNN). The original application resizes a provided image asset and detects objects within the image using a pre-trained model. A labelled image is generated with percentage weights providing the accuracy of estimation.

This system is an ideal candidate for the Fog. Fog applications that rely on this system may use the edge node to resize the original image (pre-process) so that the amount of data eventually transferred to the cloud beyond the edge is reduced. In addition, the objects can be detected at the edge to minimise communication latencies. The cloud server will send the pre-trained model to the edge that is specific to the the location that the edge resource is serving to allow the detection service to be hosted on the edge.

Application 2 - Speech-to-text conversion (PocketSphinx): PocketSphinx[10] is an open-source large vocabulary, speaker-independent continuous speech recognition engine [12]. A supplied audio file (`.wav`) is converted to a defined language in text form using a pre-trained acoustic model to determine the source and destination language for speech-to-text conversion. The integrated assets are sourced from a large scale speech repository[11]. In the Fog application, the end user-device provides a `.wav` asset to the edge container. A pre-trained acoustic model is offloaded from the cloud to the edge to facilitate text-to-speech conversion.

Application 3 - Text-audio synchronisation or forced alignment (Aeneas): The Aeneas tool automatically synchronises text and audio segments[12]. Generating synchronisation maps for a list of text fragments and an audio file containing the relevant text is referred to as forced alignment. Aeneas determines the mapping between corresponding audio segment for each text asset supplied.

In the Fog application, the end user-device transfers an audio file (`.wav`) to the edge node. Text segmentation occurs on the cloud and the text segment (`.xhtml`) is offloaded from the cloud to the edge where the forced alignment occurs. The workload on the edge is computing the synchronisation map between the text fragment and audio fragment to make it more responsive for the end-user.

Application 4 - Geo-location based online mobile game (iPokeMon): The original version of iPokeMon is a cloud-based online game that is similar to the popular virtual reality game, Pokémon Go[13]. The user device interacts with the cloud server from the beginning to the completion of the game. The Fog version employed in DeFog is the use-case that is employed for validating research on resource management at the edge of the network [31].

In the Fog version, the user creation and verification requests from the user device are made to the cloud server, after which the cloud server manager makes a request for computing services on the edge node. If this request is accepted by the edge node, then the edge manager initialises a container for the iPokeMon edge server. The cloud manager deploys the iPokeMon edge server and clones (to the edge database) location specific data and user-specific data (of those who will be connected to the edge). User data rapidly changes when the game is played. For example, the GPS coordinates of the player and the Pokémons. The local view on the edge server is updated by frequent requests sent to the server.

For experimentation, JMeter and Taurus are used to simulate players behaviour by generating synthetic workloads [31]. The user device supplies the `.jmx` file to JMeter. This allows for the automated generation of workloads for a range of concurrent users.

Application 5 - Internet-of-Things (IoT) edge gateway application (FogLAMP): This is an open source IoT application that integrates cloud storage and sensors[14]. FogLAMP interacts with endpoint sensors to collect and aggregate data (although FogLAMP allows querying on time series data originating from sensors it is not considered in this paper). A text payload containing a curl command is transferred to the edge from endpoints. The service

[8] https://github.com/qub-blesson/DeFog
[9] https://pjreddie.com/darknet/yolo/
[10] https://github.com/cmusphinx/pocketsphinx
[11] http://www.repository.voxforge1.org/downloads/SpeechCorpus
[12] https://www.readbeyond.it/aeneas/docs/
[13] http://www.pokemongo.com/
[14] https://foglamp.readthedocs.io/en/latest/

running on the edge executes the curl command by invoking a simulated API call that gathers the localised data.

Application 6 - Real-time face detection from video streams (RealFD)[15]: The original application uses an end device with an embedded video camera to capture a continuous video stream. The stream is transmitted to a cloud server where faces are detected on individual video frames using Pillow[16] and OpenCV[17].

The server comprises the following three services for detecting faces from a frame of the video: (i) *Grey-scale converter (GSC)* reduces the amount of computation done on an image that would otherwise be required if it was a colour image. The size of the stream is reduced by nearly a third, (ii) *Motion Detector (MD)* is a data filtering service that performs a condition check on successive video frames to reduce computations on similar frames (for example a security feed in a static environment, such as a home or museum). (iii) *Face Detector (FD)* is a computationally expensive service that identifies frontal faces in a video frame using machine learning.

The application can be distributed in the following four ways: (i) Cloud-only services – all the services are deployed in the cloud; (ii) Fog-based pre-processing – GSC is deployed at the edge and other services on the cloud; (iii) Fog-based data filtering – GSC and MD are offloaded to the edge and FD is deployed on the cloud; (iv) Edge-only services – all services are offloaded to the edge.

3.2 Metrics Collected by Benchmarking

Existing benchmarking methods that contain Fog conducive benchmarks either are workload specific benchmarking techniques, such as the speech-to-text benchmark[18] and TailBench [14], or are provider specific services benchmarking (AWS Greengrass and Microsoft Azure IoT Edge), such as EdgeBench[19] [6]. The motivation of DeFog is a general purpose Fog benchmarking method that can be used for a diverse range of workloads and at the same time provide both the target platform and application related benchmarks.

Three types of metrics, namely platform, application specific and from external tools are considered. These provide a bird's eye view of the target platform, including network characteristics and applications running on the platform.

(i) Target Platform Metrics: DeFog gathers platform metrics for the three deployment modes (cloud-only, edge-only, cloud-edge). A short list of these is shown in Table 2.

(ii) Fog Application Metrics: DeFog provides insight into the six application benchmarks and the metrics are outlined in Table 5. These metrics provide insight into comparing the different deployment modes and are obtained in three categories: (1) Communication metrics, include the overheads of transferring assets and data payloads during the execution of the benchmark applications, (2) Computational performance metrics, include the time taken to execute a computational task, and (3) Concurrency metrics, which quantifies the impact (performance degradation) of servicing multiple tenants on the target platform.

[15]https://github.com/qub-blesson/DYVERSE
[16]https://pillow.readthedocs.io
[17]https://opencv.org
[18]https://github.com/Picovoice/stt-benchmark
[19]https://github.com/akaanirban/edgebench

Table 2: Platform metrics collected by DeFog

Name	Description
CPU Model Name	The model name of the target platform CPU
No. 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 any assets (for example, a 34MB file for YOLOv3, which is the weights file)
Download rate	The download rate (MB/sec) of assets (for example, 200MB model file for YOLOv3)
System I/O	Speed of reading and writing in MB/sec

(iii) Metrics Gathered from External Tools: Two external tools, namely JMeter[20] and Taurus[21] are currently used by DeFog. The former is used to simulate multiple clients using a benchmark application, for example multiple users in the iPokeMon online game, The metrics obtained from JMeter is presented in Table 3. The end-to-end latency of the application when multiple users are running can be determined to identify whether the QoS of the application is met when the number of clients increases.

Table 3: Metrics collected using JMeter in DeFog

Name	Description
User/ Thread	The total concurrent users/threads
Latency	Response time latency for a specific endpoint

Taurus gathers successful response count and average response time (shown in Table 4). By simulating synthetic users metrics such as standard deviation of response time can be generated, which offers insight into the effect of outliers on median response time.

Table 4: Metrics collected using Taurus in DeFog

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 (RT)	Average response time of service
Standard Deviation of RT	Standard deviation of the response time
Average Latency	Average latency time for the return trip

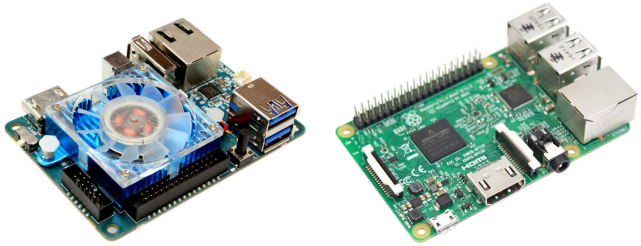
4 EXPERIMENTAL STUDIES

The setup for capturing metrics for the three deployment modes and the results obtained are presented in this section. Only a subset of metrics and results obtained are presented, given that DeFog generates an exhaustive list of metrics and results. Moreover, aggregate metrics, which is a sum of a collection of individual metrics

[20]https://jmeter.apache.org/
[21]https://gettaurus.org/

Table 5: Fog application metrics collected by DeFog

Name	Description
Execution Time	Time taken to complete a computational task (ET)
Time in Flight	Time taken to transfer a workload to the cloud or edge (T_1)
S3 Transfer Time	Time taken to upload results data to the S3 bucket (T_2)
Results Transfer Time	Time taken to return results data to the observing system (T_3)
Computation Latency	Total time taken for running the benchmark, including execution time ($RTT = T_1 + ET + T_3$)
Computation Cost	Estimated cost of the computational task ($Cost$, using the AWS pricing strategy)
Real Time Factor	Rate of speech recognition ($RTF = ET/file\_length$)
Bytes Up	Total bytes transferred to the cloud or edge
Bytes Down	Total bytes transferred from the cloud or edge
Bytes Down (cloud-edge)	Total bytes transferred from the cloud to the edge
Bytes Up per sec	Upload rate in bytes per second
Bytes Down per sec	Download rate in bytes per second
Bytes Down (cloud-edge) per sec	Download rate in bytes per second (cloud or edge) download rate
Cloud-edge Transfer Time	Time taken to transfer assets from the cloud to the edge (T_4)
Communication Latency	Total time taken for assets to be transferred throughout the benchmarking process ($CL = T_1 + T_3$)
Complete Computation Latency	Return trip time including the time taken to transfer assets from the cloud to the edge
Complete Communication Latency	Total time taken for the transfer of all payloads including the cloud asset



(a) Odroid XU4 (b) Raspberry Pi 3 Model B

Figure 5: Single board computers used as edge resources.

is considered. For example, computation latency is the sum of the time taken to transfer a service to the cloud or edge, time taken to complete executing a service, and the time taken to return output data to the observing system. It would not be possible to present and discuss the individual metrics within the scope of this paper.

4.1 Implementation and Setup

The cloud resource used is an AWS EC2 instance that is set up in the Dublin `eu-west-1` region. The platform metrics of this CPU (refer to Table 2) are an Intel Xeon E5-2676, operating at a 2.4GHz frequency, unzip time of 3.74 seconds for a file of 34MB, download rate of 9.97MB/sec for a 200MB file, and system I/O of 67.8 MB/sec.

Given that edge resources are hardware limited compared to the cloud, this research uses embedded single board computers, namely Odroid XU4 and a Raspberry Pi 3 Model B (shown in Figure 5). These have resources comparable to the compute that is available on a base station [31]. The Odroid board has 2 GB of DRAM memory, and one ARM Big.LITTLE architecture Exynos 5 Octa processor running Ubuntu 14.04 LTS. The platform metrics for the Odroid board are ARMv7 eight-core processor, operating at 2GHz, 540.65 seconds to unzip a 34MB file, download rate of 449KB/sec for a 200MB file and system I/O of 8.5MB/sec.

The Raspberry Pi has 1 GB of RAM memory, and a Quad Core 1.2GHz Broadcom BCM2837 64 bit CPU running Raspbian. The platform metrics for the Raspberry Pi are ARM v7 four-core processor, operating at 1.2GHz, 600.91 seconds to unzip a 34MB file, download rate of 404KB/sec for a 200MB file and system I/O of 4.4MB/sec.

The applications are deployed using the container technology, specifically Docker[22]. Docker 17.12.1-ce is used to automate building and deploying the application. Application specific Dockerfiles install the relevant dependencies and packages when the containers are build. Container instances are then run using the same application image. A combination of Python and bash scripting is used along with Dockerfiles to execute the benchmarks.

For each benchmark a data payload is transferred from the observing system to the edge or cloud. Fog deployment requires assets to be offloaded from the cloud to edge. Data generated on the cloud and/or edge is transferred to the S3 bucket and observing system.

YOLO, PocketSphinx, iPokeMon, Aeneas and RealFD are benchmarked across all three deployment modes. Taurus and Apache JMeter are used to simulate concurrent users. FogLAMP does not require a cloud asset, and is not used in the Fog mode, but for the cloud-only and edge-only deployment modes.

The aim of the experiments are to demonstrate the metrics that can be obtained to compare the relative computational and communication performance when the edge is leveraged. The experiments are performed on the three deployment modes (cloud-only, edge-only, and cloud-edge) for gaining insight to ***Q1*** that was posed in Section 1. The results obtained from the RealFD application (multiple services that can be moved to the edge) highlight the use of DeFog to answer ***Q2***. The experiments are carried out on two edge resources (Odroid XU4 and Raspberry Pi 3), which enhances our knowledge of the performance of the benchmark in relation to ***Q3***.

[22] https://www.docker.com/

The experiments are carried out for single and multiple users as well as when the edge resource is stressed to reflect real-world deployments. Each experiment was executed 25 times.

4.2 Results

The aim of the experimental studies is to provide insight into the benefit of a Fog system measured by communication latency and computation latency with respect to concurrent users, the Fog system stress and network stress, data transfer (uplink and downlink) rates with respect to concurrent users and Fog system and network stress, and the improvement in application response when using a Fog system against alternate architectures, such as a cloud-only or edge-only model. Therefore, the results obtained are presented as: (i) application latencies for different deployments, (ii) impact of stressing the edge on latencies, and (iii) impact of concurrent users on latencies on two cloud-edge platforms presented in Section 4.1.

4.2.1 Application latencies for different deployment modes. Figure 6 show the communication and computation round trip times for the three deployment modes, namely the cloud-only, edge-only, and cloud-edge (Fog) deployments. The standard deviation of the executions is highlighted in the observed results. In these executions, the edge resources are exclusively used by the benchmark application. The communication latency (Figure 6a) is consistently lower for all applications running on the edge when compared to the cloud. The computation latency (Figure 6b) is significantly larger for applications, such as YOLO and PocketSphinx, on the edge compared to the cloud. This is because the edge resources employed in this research are hardware limited compared to a cloud resource. The computational cost of these applications exceeds the gains in communication latencies on the edge. There is a slight increase in the latency times for the cloud-edge deployment modes when compared to the edge only deployment, which is due to the time needed to transfer assets from the cloud to the edge. Aeneas and FogLAMP show comparable and sometimes lower computational latency on the edge when compared to the cloud. The lack of performance gain for the former two applications may be due to the specific manner in which the application is partitioned.

Figure 7 shows the computation and communication latencies for different combination of services of the RealFD application (FD, MD, GSC) on the three deployment modes. For this application, the results obtained using the Odroid XU4 are presented. Communication latency in this figure is the single trip time taken (not round trip) and the computation latency is the time taken to process a single video frame. There are two potential deployment options across the cloud and edge - FD on the cloud, and MD and GSC on the edge, or alternatively FD and MD on the cloud, and GSC on the edge. There is a difference in the performance of the two Fog deployments and `DeFog` highlights this variation for the benchmark.

4.2.2 Impact of stressing edge resources on application latencies. For this experiment Gaussian workloads are simulated on the edge nodes in the cloud-edge deployment mode for YOLO, PocketSphinx, Aeneas, and RealFD and the edge-only deployment mode for FogLAMP. The motivation is to simulate a real world multi-tenant distributed system where there are competing workloads residing on the same edge under variable network conditions. The `stress`[23] package is used to stress the edge resource by simulating computations in the background. The `stress-ng`[24] package is used to stress the network. These packages use stressors to subject the computing cores and network to various levels of stress.

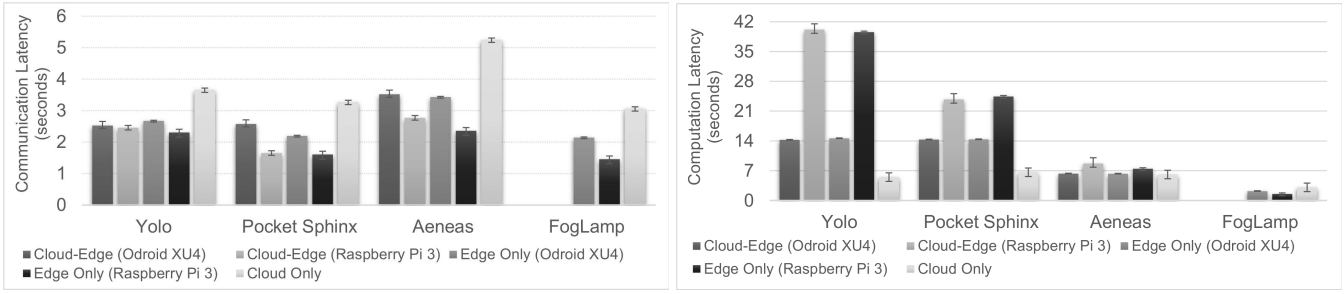
In this paper, we explicitly define minimal stress when one CPU core of the edge resource is stressed. The network is stressed by transferring a file of size 256MB at roughly 21740 bytes per second. For *low stress*, two CPU cores are stressed. For *medium* and *high stress*, three CPU cores and four CPU cores are stressed respectively. For *very high stress* all CPU cores are stressed and the RAM memory is stressed using two stressor processes.

The communication latency of benchmark applications when network bandwidth is stressed is shown in Figure 8 (for the RealFD application on Odroid XU4 is shown in Figure 10a). Applications that transfer larger amounts of data, such as Aeneas, are affected. Figure 9 shows the trend in the computation latencies of the applications when the computing cores are stressed (Figure 10b). As expected there is a significant increase in the computation latencies. Computationally less intensive applications, such as Aeneas and FogLAMP have less effect with stressed CPU cores. It is immediately inferred that applications demonstrate different trends when edge resources are stressed. For RealFD it is noted that different combination of services across the cloud and the edge have different performance. For the iPokeMon application as shown in Figure 11, the edge node is subject to similar stress as above. There is an improvement in the response latency of the cloud-edge deployment by over 7 times when compared to the cloud-only deployment when the stress as defined in this paper is very high.

4.2.3 Impact of concurrent users on application latencies. For this experiment, concurrent users (2, 5, 10, and 50) of the application benchmark are considered as shown in Figure 12. The individual requests from the application are simulated for concurrent users using JMeter. It is noted that the communication latency as shown in Figure 12a and Figure 12b increases with the number of users. This is because the time in flight and time to transfer the results also increases because with the increasing number of users the rate at which data movement occurs decreases (this is highlighted in Figure 13). Similar increases are noted for computation latency. Although the rate at which the computation latency increases is different. For example, FogLAMP has a much lower rate of increase in computation latency when compared to PocketSphinx. This is because of the nature of the computational intensity underpinning the benchmarks. The figures highlight the need for `DeFog` - to differentiate workloads that may not have a significant increase in computation or communication latencies even when there are multiple users versus those that may have the computation or communication latencies significantly affected.

Applications that have a larger execution time for a single user, are impacted the most by concurrent users, which results in larger computation latencies. For example, consider PocketSphinx as shown in Figure 14. Given the current decomposition of services for the PocketSphinx application it is evident that the cloud-edge deployment is not advantageous over the cloud-only deployment.

[23] https://people.seas.harvard.edu/~apw/stress/
[24] https://kernel.ubuntu.com/git/cking/stress-ng.git/



(a) Communication latency

(b) Computation latency

Figure 6: Latencies of applications for different deployment modes.

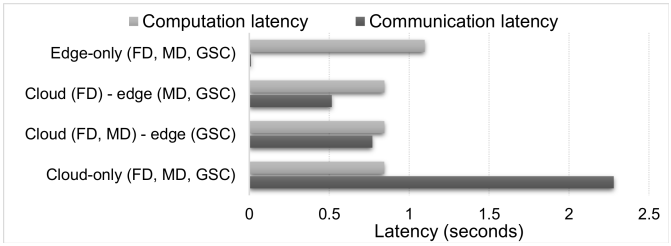
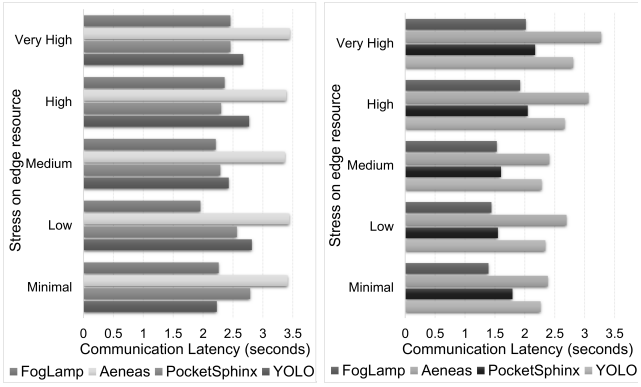


Figure 7: Latencies of RealFD for different combination of services across the cloud and the edge (using Odroid XU4).

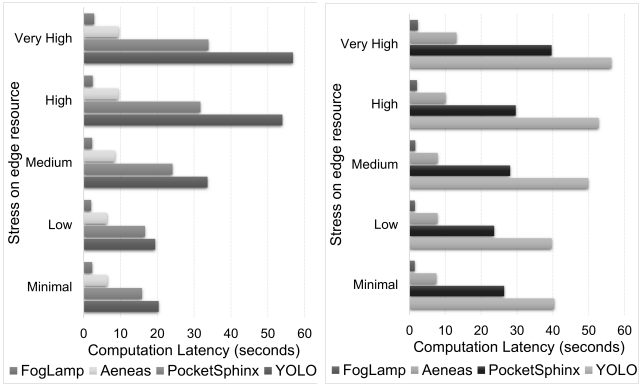


(a) On Odroid XU4

(b) On Raspberry Pi 3

Figure 8: Communication latency of benchmark applications when the edge resource is stressed.

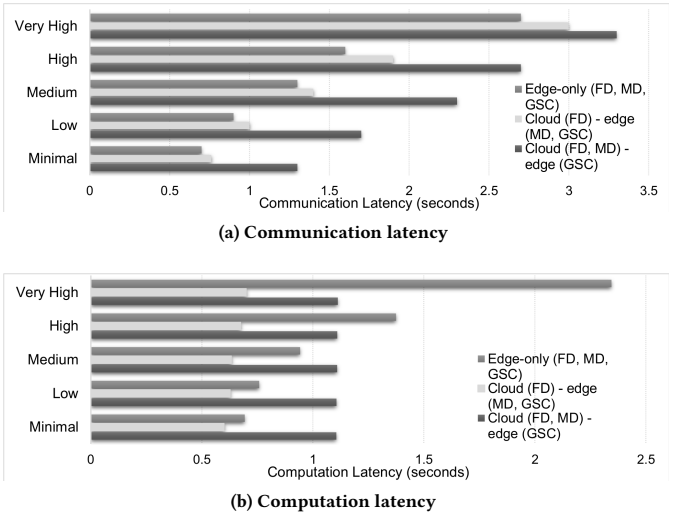
However, consider the impact of concurrent users on iPokeMon as shown in Figure 15. The cloud-edge deployment clearly has significant advantages over the cloud-only deployment. Latency spike observed for the edge nodes when there are 250 concurrent users was due to a single communication request that increased the average response time of the requests.



(a) On Odroid XU4

(b) On Raspberry Pi 3

Figure 9: Computation latency of benchmark applications when the edge resource is stressed.



(a) Communication latency

(b) Computation latency

Figure 10: Latencies of the RealFD benchmark when the edge resource is stressed.

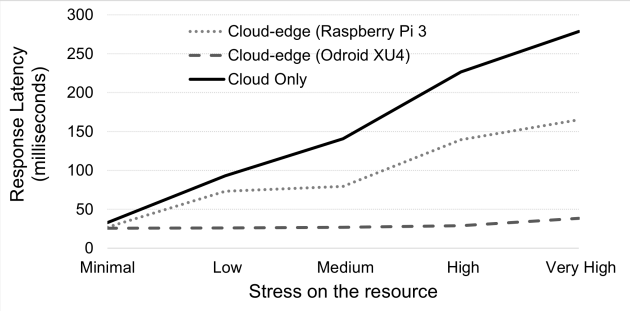


Figure 11: Impact of edge resource stress on response latency for iPokeMon.

4.2.4 Summary. The experimental results have demonstrated that the communication and computation latencies vary across the different deployments, namely cloud-only, edge-only and cloud-edge. Although the communication latency of the application may improve by using the cloud-edge deployment, there will be no overall gain if the computation latency is high on resource constrained edge nodes. Applications are noted to behave differently when the edge node is stressed, both its system and network resources. The rate at which performance degrades varies across applications. Similarly, when concurrent users request service from the same edge node, the rate at which the communication and computation latencies increase vary; some workloads exhibit significant increase in latencies where as others have a negligible increase. These observations highlight the use of DeFog to identify any performance gain in using the Fog over the cloud-only deployment (***Q1***), which services of an application would benefit from moving to the edge (***Q2***), and which deployment platforms are best suited for an application when there are multiple hardware choices (***Q3***).

5 RELATED WORK

There are multiple techniques for understanding the performance of a target platform and applications running on it. Two dominant techniques, include simulation-based and benchmarking approaches, which are applicable in the context of edge computing [11].

The simulation-based approach uses computational models to simulate the target platform. The application is modelled on the simulated platform. The application and target environment are usually abstract representations of the real application and the underlying hardware platform on which it will run is defined by a fine-grain or aggregate set of input parameters. The accuracy of running a simulation to gather performance metrics is fully dependent on the underpinning models, which may be abstract representations or fine-grain models. It is worthwhile to note that it is harder to accurately model the variability seen on a real target platform, and is more complex when modelling distributed systems, such as the Fog. Popular Fog and Edge computing based simulators include EdgeCloudSim [25], iFogSim [9] and FogExplorer[25]. Recently, emulation is also used to evaluate the performance of Fog applications [10].

The advantages of using simulation-based approaches include reproducibility of the experiments and is a cheaper solution to understand the performance of a target platform since physical hardware is not required. However, simulation approaches may not be sufficiently accurate and running fine-grain simulation models (such as discrete event simulators) can be time consuming.

On the other hand benchmarking approaches run the application on the target platform in more realistic conditions. Nonetheless, external factors that influence performance may be simulated; for example, network conditions or noise on the target platform. Since benchmarking is usually performed in the real setting, reproducibility of results depends on how transient the platform is. For example, reproducing results on the Fog, could be challenging. Nonetheless, benchmark consistency can be achieved by using technologies, such as containers. Currently no Fog benchmarks are readily available.

Accuracy and reliability of results are advantages of benchmarking since applications are executed on the target platform with fewer abstractions. However, the platform needs to be set up, which is more expensive than simulation. The application source code may also need to be modified if multiple target platforms have to be benchmarked. This paper adopts a benchmarking approach.

Existing benchmarks relevant to the discussion of this paper, include those for the cloud, end user-devices, and application specific benchmarks. Benchmarking on the cloud typically captures the performance of an application across different categories of resources, such as virtual machines (VMs) [28] and storage services [2]. Popular cloud benchmarks include Yahoo Cloud Serving Benchmark (YCSB) [5], CloudRank-D [17] and DCBench [13]. The use of containers for benchmarking in the cloud environment has advantages, such as consistency and making the benchmarking process faster and automating it [29]. However, as expected cloud benchmarking methods do not need to include techniques to capture performance of resources outside the cloud and consequently, does not need to handle the more complex dependencies that are seen in Fog application. In this paper, container-based benchmarking is extended towards taking into account the dependencies between cloud-edge services of an application as seen in the Fog.

There are benchmarking methods available for end user-devices, such as mobile [18] and IoT devices [15], and for understanding the interactions between the mobile user and the edge system [19]. IoT benchmarks focus on power consumption[26], Constrained Application Protocol (CoAP) benchmarking [15], and low power wireless network applications (such as the IoT Benchmarking Consortium's[27] D-Cube [23]). Serverless application benchmarking for CPU intensive benchmarks on edge platforms, such as Amazon Greengrass and Microsoft Azure IoT Edge are considered in literature; multiple heterogeneous cloud functions are benchmarked [6]. More application specific benchmarks have been developed. These include the TPC Express Benchmark IoT (TPCx-IoT)[28] that is based on YCSB [5] and considers data ingestion and query workloads. CAVBench is developed for benchmarking connected and autonomous vehicle applications [32].

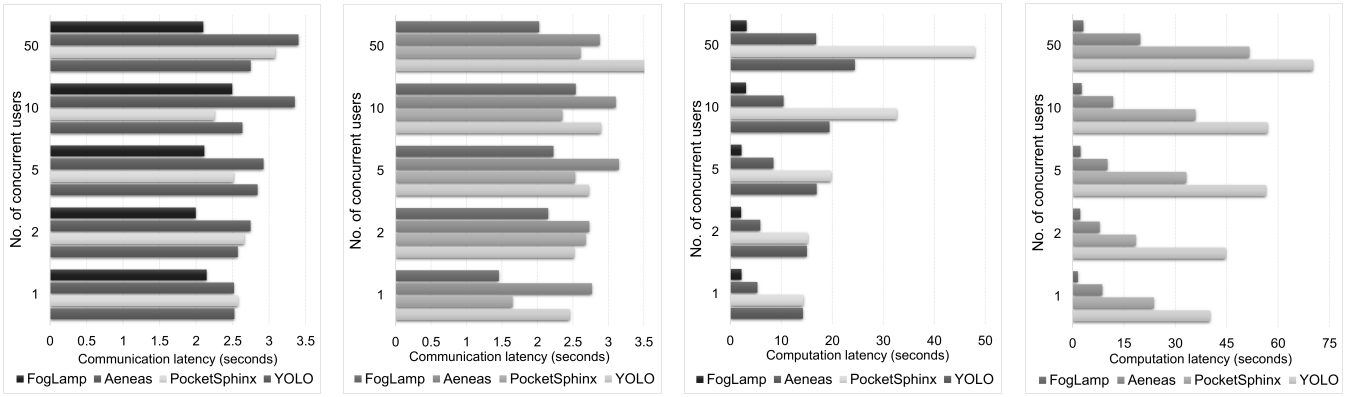
While existing approaches consider the cloud or end user-devices, benchmarking the distribution of applications across the cloud and edge has not been the focus. This paper reports a first Fog benchmarking suite.

[25] https://openfogstack.github.io/FogExplorer/

[26] https://www.eembc.org/products/

[27] https://www.iotbench.ethz.ch

[28] http://www.tpc.org/tpcx-iot/



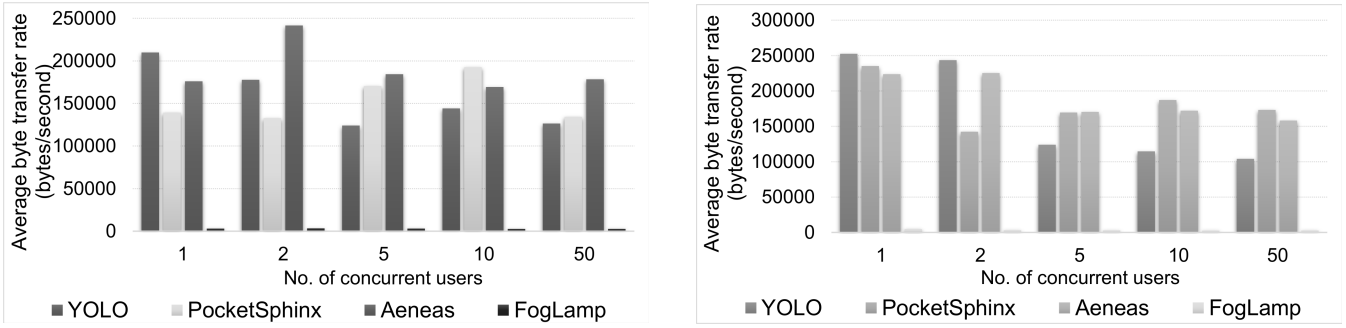
(a) Communication latency on Odroid XU4

(b) Communication latency on Raspberry Pi 3

(c) Computation latency on Odroid XU4

(d) Computation latency on Raspberry Pi 3

Figure 12: Impact of concurrent users on latency of benchmark applications.



(a) On Odroid XU4

(b) On Raspberry Pi 3

Figure 13: Impact of concurrent users on average bytes transferred.

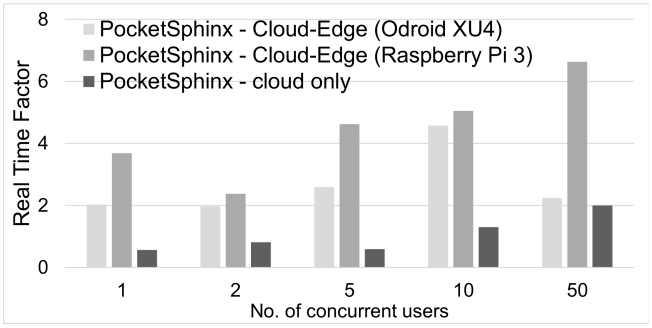


Figure 14: Impact of concurrent users on real time factor for PocketSphinx.

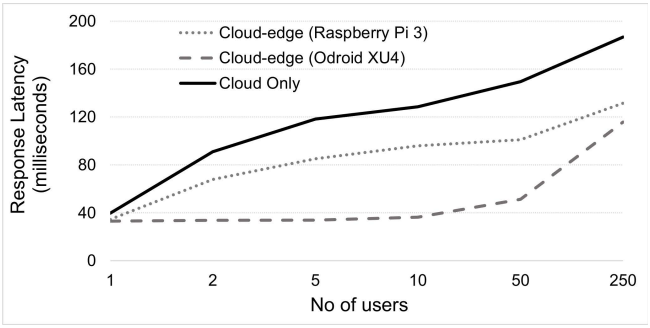


Figure 15: Impact of concurrent users on average response latency per request for iPokeMon.

6 CONCLUSIONS

There are currently no readily available Fog benchmarks to address the following questions: (***Q1***) How can the relative performance gain of using Fog platforms over the Cloud-only computing model be quantified? (***Q2***) If there are multiple services of an application that can be moved to the edge, then how can the overall performance of moving different services to the edge be quantified? (***Q3***) If there are multiple competing resources at the edge for a service, then which resource should be selected? DeFog proposed in this paper addresses the above questions.

DeFog comprises six benchmarks that can be deployed across the cloud and the edge. The current portfolio of application benchmarks in DeFog includes, deep learning-based object detection, text-to-speech conversion, text-audio forced alignment, geo-location based online mobile game, IoT edge gateway application and real-time face detection in video streams, that are deployed across a cloud-only, edge-only, and cloud-edge deployment modes. The approach taken in this paper is benchmarking actual target platforms rather than using simulation. Experimental studies demonstrate the catalogue of target platform and application related metrics captured by DeFog. The experiments also present the performance of benchmarks due to concurrent users and stress on edge resources.

The approach taken by DeFog ensures that the build and deployment of the application are automated. DeFog ensures consistency (same environment is available and dependencies are set up when comparing two different edge nodes) when benchmarking.

Limitations and Future Work: DeFog assumes that all target platforms can run containers. The container technology is limited in that processor architecture specific images are required and more hardware agnostic deployment technologies are required. It is also assumed in the implementation and experimentation that dedicated single board computer capabilities are available at the edge. Edge resources, such as routers and gateways that can be augmented with compute capabilities have not been considered. The partitioning of applications is inherently across a single layer of cloud resources and a single layer of edge resources. The current research does not explore the distribution of benchmarks across the entire cloud-edge continuum. This is because applications that can leverage the entire cloud-edge continuum are not fully understood.

The benchmarks are decomposed into individual services manually and then containerised for deployment. While DeFog can orchestrate the execution of application containers seamlessly, additional efforts are required to automatically decompose an application. A wider range of benchmarks will be included in the future to simultaneously stress all dimensions of latency, bandwidth and availability of resources, such as augmented reality applications, in a Fog environment. Additional factors, such as different ownership of cloud and edge will be considered.

The vision of DeFog is to foster community growth and development of Fog benchmarks that can be widely used.

REFERENCES

[1] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. 1991. The NAS Parallel Benchmarks - Summary and Preliminary Results. In *Proceedings of the ACM/IEEE Conference on Supercomputing*. 158–165.

[2] D. Bermbach, J. Kuhlenkamp, A. Dey, A. Ramachandran, A. Fekete, and S. Tai. 2017. BenchFoundry: A Benchmarking Framework for Cloud Storage Services. In *Proceedings of the 15th International Conference on Service-Oriented Computing*.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. 2012. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. 13–16.

[4] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amosand G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. Klatzky, D. Siewiorek, and M. Satyanarayanan. 2017. An Empirical Study of Latency in an Emerging Class of Edge Computing Applications for Wearable Cognitive Assistance. In *Proceedings of the 2nd ACM/IEEE Symposium on Edge Computing*. Article 14, 14:1–14:14 pages.

[5] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*. 143–154.

[6] A. Das, S. Patterson, and M. P. Wittie. 2018. EdgeBench: Benchmarking Edge Computing Platforms. In *Proceedings of the 4th International Workshop on Serverless Computing*.

[7] J. J. Dongarra, P. Luszczek, and A. Petitet. 2003. The LINPACK Benchmark: Past, Present and Future. *Concurrency and Computation: Practice and Experience* 15, 9 (2003), 803–820.

[8] D. Fesehaye, Y. Gao, K. Nahrstedt, and G. Wang. 2012. Impact of Cloudlets on Interactive Mobile Cloud Applications. In *16th IEEE International Enterprise Distributed Object Computing Conference*. 123–132.

[9] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya. 2017. iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in the Internet of Things, Edge and Fog Computing Environments. *Software: Practice and Experience* 47, 9 (2017), 1275–1296.

[10] J. Hasenburg, M. Grambow, E. Grunewald, S. Huk, and D. Bermbach. 2019. MockFog: Emulating Fog Computing Infrastructure in the Cloud. In *IEEE International Conference on Fog Computing*. 192–201.

[11] C. H. Hong and B. Varghese. 2019. Resource Management in Fog/Edge Computing: A Survey on Architectures, Infrastructure, and Algorithms. *Comput. Surveys* (2019).

[12] D. Huggins-Daines, M. Kumar, A. Chan, A. W. Black, M. Ravishankar, and A. I. Rudnicky. 2006. Pocketsphinx: A Free, Real-Time Continuous Speech Recognition System for Hand-Held Devices. In *Proceedings of the IEEE International Conference on Acoustics Speech and Signal Processing*.

[13] Z. Jia, L. Wang, J. Zhan, L. Zhang, and C. Luo. 2013. Characterizing Data Analysis Workloads in Data Centers. In *IEEE International Symposium on Workload Characterization*. 66–76.

[14] H. Kasture and D. Sanchez. 2016. Tailbench: A Benchmark Suite and Evaluation Methodology for Latency-critical Applications. In *IEEE International Symposium on Workload Characterization*.

[15] C. P. Kruger and G. P. Hancke. 2014. Benchmarking Internet of Things Devices. In *12th IEEE International Conference on Industrial Informatics (INDIN)*. 611–616.

[16] Z. Li, L. O'Brien, H. Zhang, and R. Cai. 2012. On a Catalogue of Metrics for Evaluating Commercial Cloud Services. In *ACM/IEEE 13th International Conference on Grid Computing*. 164–173.

[17] C. Luo, J. Zhan, Z. Jia, L. Wang, G. Lu, L. Zhang, C.-Z. Xu, and N. Sun. 2012. CloudRank-D: Benchmarking and Ranking Cloud Computing Systems for Data Processing Applications. *Frontiers of Computer Science* 6, 4 (2012), 347–362.

[18] N. Z. Naqvi, T. Vansteenkiste-Muylle, and Y. Berbers. 2015. Benchmarking Leading-edge Mobile Devices for Data-intensive Distributed Mobile Cloud Applications. In *IEEE Symposium on Computers and Communication*. 50–57.

[19] M. Olguin, J. Wang, M. Satyanarayanan, and J. Gross. 2018. Scaling on the Edge – A Benchmarking Suite for Human-in-the-Loop Applications. In *Proceedings of the IEEE/ACM Symposium on Edge Computing*.

[20] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. In *IEEE Conference on Computer Vision and Pattern Recognition*. 779–788.

[21] J. Redmon and A. Farhadi. 2018. YOLOv3: An Incremental Improvement. *CoRR* abs/1804.02767 (2018). arXiv:1804.02767 http://arxiv.org/abs/1804.02767

[22] M. Satyanarayanan. 2017. The Emergence of Edge Computing. *Computer* 50, 1 (2017), 30–39.

[23] M. Schuss, C. A. Boano, and K. Romer. 2018. Moving Beyond Competitions: Extending D-Cube to Seamlessly Benchmark Low-Power Wireless Systems. In *IEEE Workshop on Benchmarking Cyber-Physical Networks and Systems*. 30–35.

[24] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.

[25] C. Sonmez, A. Ozgovde, and C. Ersoy. 2017. EdgeCloudSim: An Environment for Performance Evaluation of Edge Computing Systems. In *International Conference on Fog and Mobile Edge Computing*. 39–44.

[26] B. Varghese, O. Akgun, I. Miguel, L. Thai, and A. Barker. 2019. Cloud Benchmarking for Maximising Performance of Scientific Applications. *IEEE Transactions on Cloud Computing* 7, 1 (2019), 170–182.

[27] B. Varghese and R. Buyya. 2018. Next Generation Cloud Computing: New Trends and Research Directions. *Future Generation Computer Systems* 79, 3 (2018), 849–861.

[28] B. Varghese, L. T. Subba, L. Thai, and A. Barker. 2016. Container-Based Cloud Virtual Machine Benchmarking. In *IEEE International Conference on Cloud Engineering*. 192–201.

[29] B. Varghese, L. T. Subba, L. Thai, and A. Barker. 2016. DocLite: A Docker-Based Lightweight Cloud Benchmarking Tool. In *IEEE/ACM 16th International Symposium on Cluster, Cloud and Grid Computing*. 213–222.

[30] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos. 2016. Challenges and Opportunities in Edge Computing. In *Proceedings of the IEEE International Conference on Smart Cloud*. 20–26.

[31] N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos. 2017. ENORM: A Framework for Edge Node Resource Management. *IEEE Transactions on Services Computing* (2017). https://doi.org/10.1109/TSC.2017.2753775

[32] Y. Wang, S. Liu, X. Wu, and W. Shi. 2018. CAVBench: A Benchmark Suite for Connected and Autonomous Vehicles. In *Proceedings of the 3rd ACM/IEEE Symposium on Edge Computing*.