Computer Networks 181 (2020) 107526

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Characterizing task completion latencies in multi-point multi-quality fog computing systems

Maria Gorlatova [a,*], Hazer Inaltekin [b], Mung Chiang [c]

[a] Department of Electrical and Computer Engineering at Duke University, Durham, NC 27708, USA
[b] School of Engineering, Macquarie University, North Ryde, NSW 2109, Australia
[c] School of Electrical and Computer Engineering, Purdue University, West Lafayette,, IN 47907, USA

ARTICLE INFO

Keywords:
Fog computing
Latency
Measurement study
Smart gateway
Serverless computing

ABSTRACT

Fog computing, which distributes computing resources to multiple locations between the Internet of Things (IoT) devices and the cloud, is attracting considerable attention from academia and industry. Yet, despite the excitement about the potential of fog computing, few comprehensive studies quantitatively characterizing the properties of fog computing architectures have been conducted. In this paper we examine the statistical properties of fog computing *task completion latencies*, which are important to understand to develop algorithms that match IoT nodes' tasks with the best execution points within the fog computing substrate. Towards characterizing task completion latencies, we developed and deployed a set of benchmarks in 6 different locations, which included local nodes of different grades, conventional cloud computing services in two different regions, and Amazon Web Services (AWS) and Microsoft Azure serverless computing options. Using the developed infrastructure, we conducted a series of targeted experiments with a node invoking our benchmarks from different locations and in different conditions. The empirical study elucidated several important properties of task execution latencies, including latency variation across different execution points and execution options, and stability with respect to time. The study also demonstrated important properties of serverless execution options, and showed that statistical structure of computing latencies can be accurately characterized based on a small number (only 10–50) of latency samples. The complete measurement set we have captured as part of this study is publicly available.

1. Introduction

Fog computing is an emerging paradigm in which computing, storage, networking, and control are placed at multiple locations between the endpoint devices and the cloud [1–3]. Fog computing is receiving increasing attention from industry and academia alike [1,4–12] due in part to its potential for enabling advances in the Internet of Things (IoT) applications [2]. Compared to centralized cloud solutions, fog computing can provide task execution points with different properties, and can offer significantly lower IoT task execution latencies (as fog execution points can be located closer to the end users) [13,14]. Multiple lines of work have recently considered the development of task scheduling and resource allocation algorithms for the scenarios where different IoT nodes' tasks can be executed in multiple locations within a fog computing substrate [8–10,15–20]. However, little attention has been paid to the empirical properties of task completion latencies in such settings, and to the ease – or difficulty – of obtaining characterizations of latency properties of different in-fog execution points. Our work addresses a part of this gap, as we describe below.

In this work we focus on characterizing statistical properties of task completion latencies in fog computing, for systems with heterogeneous, geographically distributed task execution points, as shown in Fig. 1. In the settings we examine, a locally placed gateway is deciding where, within the fog substrate that includes both local and cloud task execution points, it should execute the tasks of the IoT nodes. Gateway-based IoT architectures are currently a *de-facto* standard: many modern IoT architectures connect low-end IoT devices to the Internet via gateways placed nearby, within the range of a direct WiFi or Bluetooth connection. At different fog execution points, the tasks can be executed with different *execution options*, e.g., with different quality levels. Multi-quality approaches are starting to be actively explored; many different ones have been proposed recently [19,21–24]. They are particularly attractive for heterogeneous fog systems with many IoT nodes, where the range of latency requirements of different nodes may necessitate supporting a wide range of task execution options.

* Corresponding author.
E-mail addresses: maria.gorlatova@duke.edu (M. Gorlatova), hazer.inaltekin@mq.edu.au (H. Inaltekin), chiang@purdue.edu (M. Chiang).

https://doi.org/10.1016/j.comnet.2020.107526
Received 19 December 2019; Received in revised form 26 August 2020; Accepted 27 August 2020
Available online 1 September 2020

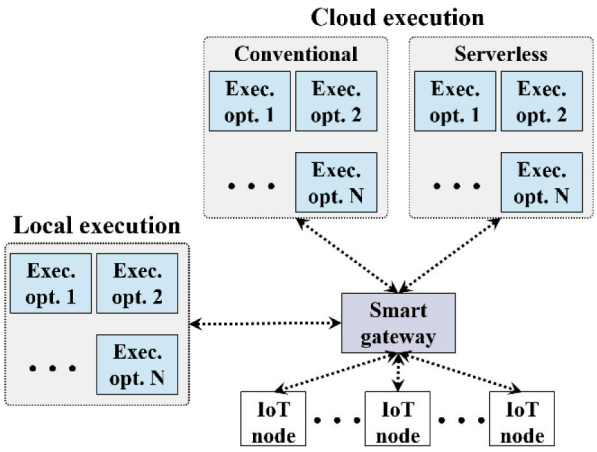


Fig. 1. A multi-execution-point multi-quality fog computing system serving IoT nodes. IoT nodes' tasks can be executed in multiple locations within the fog computing substrate: on local devices, as well as on the cloud, with both conventional and serverless cloud computing options available.

Table 1
Example computing capabilities of different fog computing execution point categories.

Fog computing point	Device or service	CPU cores: number, frequency	Additional computing capabilities
Local server	PowerEdge R930 [31]	4–24 cores @ 2.2–3.5 GHz	GPUs
Cloud, conventional	AWS C2 [32]	1–128 cores @ 2.3–3.3 GHz	GPUs, FPGAs
Cloud, serverless	AWS Lambda [33]	Fractional parts of C2 cores	None

We focus on *responsive* IoT applications, in which IoT nodes rely on fog computing for timely responses to their requests — as opposed to, for example, fog-based execution of data analytics tasks without explicit deadlines. The need for timely responses arises in both human-facing and machine-to-machine applications, e.g., in gaming, augmented and virtual reality [22,25], user guidance and feedback [14,26,27], motion control in cars, autonomous drones [28], and industrial machinery [29,30], and control of parameters in cyber–physical systems. Responsive applications are considered to be one of the primary use cases for fog computing [1,4]. In many of these systems, both the average task completion latency and the associated broader latency-related Quality of Service parameters (e.g., expected jitter, 95th and 99th latency percentiles) are important for providing the best experience. Correspondingly, in this work we consider not only the mean latencies for different execution points and computing options, but also task completion latencies as random variables, to identify their statistical structure in detail.

Below we first provide an overview of several classes of task execution points in fog computing (Section 1.1), then describe our contributions (Section 1.2) and paper organization (Section 1.3).

1.1. Categories of execution points in fog computing

Execution points present in fog computing systems have different properties, as we describe below. Several representative options of different fog node categories are shown in Table 1.

Local servers: Despite the rising popularity of cloud computing services, on-premise computing remains an essential feature of many organizations, including universities. Ranging from small stand-alone machines to full on-site datacenters, local computing infrastructure offers computing options in close proximity to the local IoT nodes, for example, in-building, or on the same campus [34]. We included a local server-grade node in our study, and a local consumer-grade node for comparison.

Cloud, conventional: Traditional cloud services' computing capabilities are located in one of several global datacenters (e.g., AWS and Google Cloud offer services in 24 and 23 geographical regions worldwide, respectively). Cloud computing has been commercially available for over a decade. As part of the cloud services, multiple options with different underlying hardware and different performance guarantees are currently available [32]. We included two AWS execution points, located in different regions, in our study.

Cloud, serverless: *Serverless* cloud computing allows running computing using cloud datacenter resources without managing specific computing instances. That is, rather than being provisioned ahead of time, individual functions are *instantiated on demand* when they are requested. First introduced by Amazon in 2014, serverless computing is now offered by all major cloud providers [33,35–37], and is actively promoted specifically for IoT applications [33,36]. We included two AWS-based serverless execution points and one Microsoft Azure-based serverless execution point in our study. To the best of our knowledge, we are the first to closely examine the properties of serverless computing as an execution option for latency-sensitive tasks in fog computing.

1.2. Our contributions

In this paper, we empirically examine the properties of task execution latencies in heterogeneous fog computing systems. Towards this examination, we developed and deployed a set of benchmarks in 6 different locations, which included local nodes of different grades, conventional cloud computing services in two different regions, and serverless computing options from both Amazon Web Services (AWS) and Microsoft Azure. Using the developed infrastructure, we conducted a series of targeted experiments with a computing node invoking these benchmarks from different locations and in different conditions, including from 10 different environments on a trip between different geographic regions, and continuously for 30 days with different inter-invocation times. Altogether we obtained over 1000 h of measurements. The contributions of our empirical study are as follows:

- We elucidate several important properties of task execution latencies, including their heterogeneity across different locations and stability with respect to time. We also demonstrate that latency characterizations of specific execution options can be obtained with as few as 10–50 latency samples.
- We demonstrate important properties of serverless execution options. For instance, a gateway serving multiple IoT nodes has the potential to lower the response latency if it assigns multiple nodes' tasks to the same serverless execution point.
- We provide a set of guidelines for smart gateway-based latency characterizations, predictions, and task assignments. Our guidelines include, for example, the ability to treat serverless execution points as having infinite capacities, the need to separately characterize latencies of different execution options on the same execution point, and the possibility of using Generalized Extreme Value (G.E.V.) distributions to describe latency distributions of fog execution options.

The complete measurement set we collected as part of this study is available at [38].

1.3. Paper organization

The rest of the paper is organized as follows. Section 2 describes the related work. Section 3 presents our experimental settings. Sections 4 and 5 present the results of the examinations of different types of fog computing points: conventional execution, local and remote (Section 4), and serverless execution (Section 5). Section 6 demonstrates

Table 2
Existing relevant empirical task completion latency examinations.

#	Public.	Year	Fog/ cloud	Public dataset	Stability w.r.t. time	Serverless execution
1	Chen *et al.* [14]	2017	Cloud and edge	No	No	No
2	Zheng *et al.* [52]	2014	Cloud	Yes	Yes	No

how task completion latencies can be modeled as random variables. Section 7 demonstrates that these models can be reliably obtained from a limited number of task completion latency samples, while Section 8 comments on the stability of the resulting task completion latency distributions with respect to time. Section 9 concludes the paper.

2. Related work

Our study is motivated by multiple lines of work on IoT nodes' task assignments to execution points in fog computing systems [8–10,16–20,39–43]. Due to fog systems' potential to reduce task execution latency compared to cloud-based systems, in many cases task allocation objectives involve *optimizing task completion latencies*, or optimizing other parameters subject to latency constraints [8,9,16–20,39]. For example, [8] aims to minimize the total weighted response time of all tasks, and [39] aims to minimize the maximal weighted cost of delay and energy consumption among all users. In this paper we aim to provide experimental insights and data that inform such "task placement" algorithms.

Networking and communication latencies have long been studied, in different settings and from different perspectives (e.g., WiFi latency [44], latency associated with accessing different cloud services [45]). These studies focus on network parameters alone, while our study includes latencies associated with communication, service initialization, and task execution, which all represent important elements of the delays experienced by IoT nodes that rely on in-fog services for task execution.

Empirical observations of task completion latencies in edge and fog computing have been made in [46–48], among others. Such studies typically focus on first-order latency characterizations, often in the context of a specific application optimized for in-fog execution. First-order observations about latency variations in serverless computing have recently been made in [49–51]. By contrast, we focus on task completion latency models, variability, stability, and other complex properties. *To the best of our knowledge, our work is the first to consider task latency characterizations from the point of view of a smart gateway, and the first to provide corresponding guidelines.*

Relevant latency measurements have also been conducted for cloud gaming [13,53,54]. While relevant, gaming settings do not fully correspond to the settings where IoT nodes are requesting services from multiple heterogeneous execution points in a fog computing platform.

Two papers that are closest to ours are [14], an empirical study of latency in edge and cloud settings for wearable cognitive assistance, and [52], an empirical examination of the latency and the quality of service of real-world web services (see Table 2). Similar to our work, [14,52] examine task completion latencies that include communication and task execution delays. [14] considers the effects of edge and cloud-based execution, last-hop connectivity (4G LTE and WiFi), mobile hardware, and edge-based accelerators. Our work complements [14] by examining other factors, such as task latency modeling as a random variable, its stability with respect to time, and latency of serverless execution, and by providing a public dataset of our measurements. [52] examines availability, latency, and throughput of real-world web services monitored on a large scale over a long period of time, and shares the associated dataset with the broader research community. Our work complements [52] by focusing specifically on fog-based heterogeneous services for IoT nodes, rather than general user-facing web services. As part of our examination, we explicitly consider multiple elements not examined in [52], such as different types of platforms (traditional and serverless; local and remote), and multiple execution quality levels.

Table 3
Fog computing execution points we deployed.

#	Computing node	Locations	CPU specifications
1	Local server-grade node	Campus	Intel Xeon E5-2609 4@2.4 GHz
2	Local consumer-grade node	Residential location	Intel Atom 230 1@1.60 GHz
3	AWS EC2 t2.micro instance	US East (North Virginia), US West (Oregon)	Intel Xeon E5-2676 1@2.40 GHz
4	AWS Lambda	US East (North Virginia), US West (Oregon)	Various, from AWS EC2 pool
5	Microsoft Azure serverless functions	US East (North Virginia)	Various, from Microsoft Azure pool

Table 4
Benchmarks we deployed.

#	Task	Description	Execution options
1	π Calculations (PIC)	Calculate π to many decimal places via Leibnitz approximation.	5000–500,000 iterations
2	Process Stored File (PSF)	Calculate statistics on a dataset read from node memory.	500–50,000 lines read
3	File send-and-process (FSP)	Calculate statistics on a dataset sent to the node.	500–10,000 lines transmitted

3. Measurement settings

We deployed a set of custom benchmarks on fog computing points listed in Table 3. These points included both local nodes and cloud services with both conventional and serverless cloud execution options.

The benchmarks we developed (in Python) and deployed on the above-listed nodes are summarized in Table 4. These benchmarks stress different aspects of a computing system. The *π Calculations (PIC)* task stresses the CPU (a similar Super π procedure is used to characterize the performance of conventional and overclocked CPUs [55]). The *Process Stored File (PSF)* task stresses memory access. The *File Send-and-Process (FSP)* task stresses the communications. The benchmarks do not correspond to one specific application, but are rather representative of CPU-limited, memory-limited, and communications-limited IoT nodes' tasks. The benchmarks are representative of low-complexity tasks that could be requested by IoT nodes. The benchmarks are executed with a range of execution options shown in Table 4, corresponding to the envisioned multi-quality execution in future fog computing systems. While the three tasks can be executed with much larger execution options (leading to longer execution times), we specifically selected smaller execution options to correspond to *responsive* IoT applications, in which IoT nodes rely on fog computing for timely responses to their requests.

On conventional computing points we deployed the benchmarks over the popular Flask web services framework [56], with communication over the HTTP request–response protocol. The codebase deployed in all conventional computing nodes was identical. For serverless execution, which relies on proprietary deployment agents, we adapted

Table 5
ICMP ping times for fog nodes accessed from an on-campus location and a nearby residential location.

	ICMP ping latency, ms, for different fog nodes				
	Measurement	Campus	Residential	US East	US West
On-campus node	Median	2.3	16	8.7	70.4
	90th percentile	4.4	21.4	9.6	71.4
Residential node	Median	15.2	0.9	18.7	86.8
	90th percentile	21.1	1.4	23.7	92.1

our code slightly to adhere to the required serverless input and output handling.

As *local execution points*, we selected a server-grade option (#1 in Table 3) with an Intel Xeon CPU, and a consumer-grade option (#2 in Table 3) with an Intel Atom CPU. These execution points were not fully dedicated to our purposes, but were lightly loaded. We included a server-grade and a consumer electronics-grade local execution points in our study, to correspond to various local execution options that could potentially be available in fog computing architectures.

As *cloud execution points*, we used two AWS datacenters, one in our geographic region (US East, North Virginia, ≈300 miles away from our campus), and one that is far away (US West, Oregon, >2500 miles from campus). For conventional cloud-based execution we used EC2 t2.micro instances [32]. We also included serverless AWS (AWS Lambda) [33] and Microsoft Azure options [36] in our experiments (#4 and #5 in Table 3). We ran serverless Microsoft Azure functions in the US East Microsoft datacenter, and serverless AWS options in the same US East and US West computing centers as the conventional AWS cloud services. These commercial platforms share resources between hundreds of thousands of clients, and offer no visibility into their CPU, memory, and network utilization rates. Despite extensive resource multiplexing, which can be associated with fluctuations in resource availability, we have observed that task latency statistics are relatively stable with respect to time, as we describe in Section 8.

The majority of our experiments were conducted with a wirelessly connected laptop, which represented a smart gateway. We used only one laptop at a time, thus the overall service invocation rate was the same as the laptop's task invocation rate. The services were not loaded with other nodes' requests; service queuing delays were not a factor in our latency characterizations. The laptop called the deployed services from either an on-campus location or a close-by residential location (<15 miles from campus). The ICMP ping response times for the different fog nodes accessed from these location are shown in Table 5. As expected, at each location a local fog node can be reached faster than the closest cloud node (2.3 ms vs. 8.7 ms for the on-campus location, 0.9 ms vs. 18.7 ms for the residential one). We also performed several experiments with other locations (e.g., the experiments described in Section 4), and several with the local nodes connected via Ethernet (e.g., the experiments described in Section 8.2).

We observed that *connections with the cloud are notably faster on-campus than off-campus*. For example, as listed in Table 5, we measured ICMP ping delays as 8.7 ms vs. 18.7 ms for the US East region, and 70.4 ms vs. 86.8 ms for the US West region. Unusually low latency of campus-to-cloud connections has been noted before [47]; we observed it on both Princeton University and Duke University campuses. This observation suggests that *on-campus experiments do not fully capture the experience of the majority of the users of fog computing architectures*, and suggests the need to include off-campus locations in experimental examinations of fog computing architectures.

We measured task completion latencies by measuring the round-trip time between a laptop issuing task execution request and receiving a result transmitted by the task execution point. This method corresponds to measuring latency in its entirety. For local servers and conventional cloud servers, the incurred delays include laptop-to-server communications and server-based task execution components. For serverless execution, in which functions are instantiated on demand when they are requested, the incurred delays also include server-based service initialization components. To partially decouple the impacts of local and remote latency impacts, we interleaved the invocations of benchmarks on different execution points, local and remote. In particular, this approach helped us distinguish between the WiFi-associated local connectivity disruptions, which affect communications to all execution points, and the conditions affecting individual execution points. In different parts of this research we evaluated a range of task completion latency properties, including latency magnitude for different settings, latency distributions, and latency stability with respect to time.

Table 6
Median and 95th percentile task completion latencies for the PIC benchmark requested by a laptop located on campus. Each data point in this table is based on over 1200 data samples.

Execution point	Median latency (s)			95th percentile latency (s)		
	5000 iter.	50,000 iter.	500,000 iter.	5,000 iter.	50,000 iter.	500,000 iter.
Server	0.02	0.06	0.47	0.04	0.08	0.64
Cons. node	0.08	0.38	3.0	0.12	0.61	3.5
AWS EC2, US East	0.03	0.05	0.37	0.04	0.07	0.52
AWS EC2, US West	0.15	0.18	0.48	0.16	0.19	0.62
AWS Lambda, US East	0.46	0.52	1.19	0.57	0.65	1.37
AWS Lambda, US West	0.72	0.79	1.05	0.86	0.92	1.27

Altogether we collected over 1000 h of experimental data, which included several multi-day experiments, some lasting as long as 30 days. Example summaries of the measured task completion latencies, for the PIC benchmark executed on different fog nodes with different execution options, are shown in Table 6. Each value in this table is based on more than 1200 individual data samples. We can see, for example, that for this task, the median completion latency is the smallest on a local server for the least complex execution option, but is the smallest on the closest non-serverless cloud server, AWS EC2 US East, for the two more complex execution options. This is intuitive: for small tasks, the latency is dominated by the communication time, which is smaller for local nodes, while for larger tasks, it is dominated by the execution time, which can be smaller for more capable cloud nodes. We note, however, that these tradeoffs can be different for consumer-grade and server-grade local nodes. For this task, for example, there are no conditions under which the execution on a local consumer-grade node is faster than execution on the cloud. Finally, we also observe that serverless AWS Lambda execution is significantly longer than non-serverless options.

4. Role of geography and local conditions

In this section we comment on geography-related task completion latency properties in multi-point fog computing systems.

It is well-known that the communication delay differs for different datacenters, with the delay dependent on the distance signals need to travel. To further understand the variability of task completion latencies across different edge conditions and locations, we did a direct comparison of latency distributions for multiple PIC task executions in US East and US West datacenters, for a laptop carried on a trip from a location on the Eastern side of the US (Durham, NC) to a location to the Western side of the US (Seattle, WA). The experiment included 16.3 h of measurements collected in 10 different environments; the laptop was accessing the benchmarks over WiFi and cloud connectivity available in a given environment. Table 7 lists the environments we captured in these experiments, and summarizes the key statistics of latency

Table 7
The statistics of completion latencies captured, for execution points in AWS EC2 US East and US West datacenters, for 10 different environments.

#	Location	Execution point: AWS EC2, US East	Execution point: AWS EC2, US West
1	Duke University campus, Durham, NC	m = 0.34, sp = 0.1 (N = 721)	m = 0.48, sp = 0.1 (N = 293)
2	Residential settings, Durham, NC	m = 0.53, sp = 0.17 (N = 8627)	m = 0.49, sp = 0.43 (N = 2814)
3	Raleigh–Durham Airport (RDU), Durham, NC	m = 0.40, sp = 0.11 (N = 303)	m = 0.50, sp = 0.11 (N = 91)
4	Flight from Durham, NC to Houston, TX	m = 0.56, sp = 0.42 (N = 721)	m = 0.66, sp = 0.42 (N = 254)
5	Houston Airport (HOU), Houston, TX	m = 0.43, sp = 0.43 (N = 624)	m = 0.55, sp = 0.99 (N = 177)
6	Flight from Houston, TX to Seattle, WA	m = 1.92, sp = 0.39 (N = 1768)	m = 1.84, sp = 0.39 (N = 589)
7	Hotel in Seattle, WA	m = 0.44, sp = 0.07 (N = 8510)	m = 0.33, sp = 0.07 (N = 2906)
8	Corporate campus in Seattle, WA	m = 0.51, sp = 0.20 (N = 1504)	m = 0.38, sp = 0.20 (N = 501)
9	Seattle airport (SEA), Seattle, WA	m = 0.55, sp = 0.32 (N = 766)	m = 0.42, sp = 0.30 (N = 249)
10	Flight from Seattle, WA to Chicago, IL	m = 4.40, sp = 0.77 (N = 426)	m = 4.28, sp = 0.78 (N = 161)

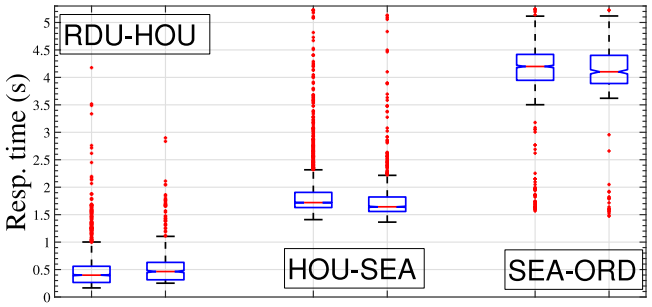


Fig. 2. Task execution latencies recorded with task invocations requested via in-flight WiFi, on 3 different flights, RDU-HOU, HOU-SEA, and SEA-ORD, for the task execution in two datacenters: US East (the left boxplot for each of the flights), and US West (the right boxplot for each of the flights). In-flight, task execution latencies are substantially higher than the latencies in other conditions. Task execution latency differences between the US East and the US West datacenters are small in comparison with the overall in-flight task execution latencies.

distributions for the case of the PIC execution option with 500,000 iterations. The table shows the median latencies m and the spans sp between the 10th and the 90th percentiles of latency distribution, for the N task latency samples we captured in each of the experiments.

Comparing task completion latencies in US East and US West execution points, as experienced from the different locations, we observe, as expected and as recorded before (e.g., in [47]), that the median task completion latency is usually smaller when the task is executed in a nearby datacenter. The only exception we noticed is location 2 in Table 7, for which the execution in the nearby datacenter is slightly slower than the execution in the remote datacenter; we hypothesize that this is associated with a slow-down in the execution in the nearby datacenter. For the conditions we examined, the typical differences in median latencies between the US East and the US West datacenters were in the range of 100–140 ms.

As part of this experiment, we collected data on three different flights, where we were using in-flight WiFi: Raleigh, NC to Houston, TX (RDU-HOU), Houston, TX to Seattle, WA (HOU-SEA), and Seattle, WA to Chicago, IL (SEA-ORD). The key latency statistics for these flights are summarized in Table 7. Latency distribution boxplots are shown in Fig. 2. In-flight WiFi is notoriously slow, which we readily observed in our experiments: e.g., as shown in Table 7, in the in-flight conditions, the median latency values are the highest of all environments we examined. Coincidentally, our experience also showcased the difference between "basic" and "high-speed" in-flight WiFi. On the RDU-HOU flight, the median task completion latency was only 160 ms higher than the median latency recorded in the flights' departure airport, RDU (a 40% increase). On the other two flights, the median latencies were 1.5 s and 3.9 s higher than the median latencies in the departure airports, *>300% and >700% increases*, correspondingly. We deduced that the RDU-HOU flight potentially has happened to be among the 7.2% seat-miles of flights that currently offer higher-quality WiFi [57]. Task completion latency distributions were also somewhat less stable in in-flight conditions, as we elaborate on in Section 8. Comparing task execution latencies between the US East and the US West datacenters, we observe that in the in-flight conditions, task execution latency differences between the datacenters are small in comparison with the overall in-flight task execution latencies: namely, 100 ms (RDU-HOU), 80 ms (HOU-SEA), and 120 ms (SEA-ORD).

5. Serverless execution

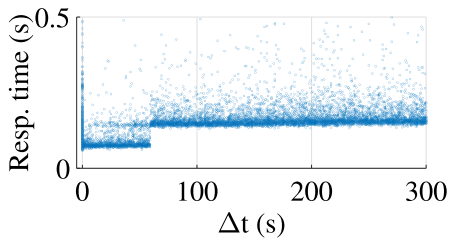
In this section we examine latencies associated with *serverless* task execution, in which functions are instantiated on demand when they are requested. Serverless execution is widely used for web applications, and is promoted specifically for the Internet of Things and fog computing [1,49,58]. Below, we first comment on unique attributes of serverless computing (Section 5.1), then describe the results of our task execution latency characterizations (Section 5.2), and comment on the implications of serverless properties for smart gateway-based latency characterizations (Section 5.3) and decision-making (Section 5.4). The characterizations were conducted with both AWS Lambda and Microsoft Azure serverless functions, as described in Section 3. The measurements we collected for the analysis presented in this section are available at [38]. We are unaware of existing public datasets that capture the serverless execution properties we describe below.

5.1. Autoscaling in serverless computing

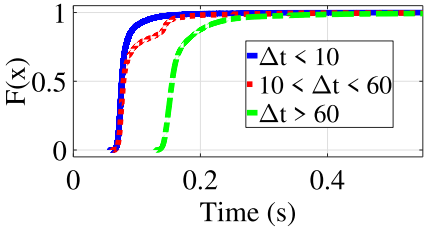
One of the core advantages of serverless computing is the so-called *autoscaling*: unlike traditional cloud computing resources, which need to be created ahead of time, serverless instances are spun up and down automatically based on user demand [59]. This saves application developers and administrators from worrying about creating the right number of processes for the users.[1]

The downsides of serverless computing are some limitations in languages and functionality, the need to adapt programs to serverless APIs (which we had to do for our benchmarking experiments), and the associated complexity of porting serverless solutions from one provider to another [60]. An additional downside of autoscaling is the complex latency characteristics of serverless execution, as we describe below.

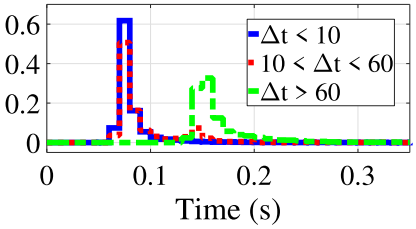
[1] Throughout our benchmarking efforts, we appreciated the autoscaling property of serverless computing: we had to keep track of whether we instantiated the conventional services, but did not have to worry about serverless ones.



(a) Task completion latency vs. Δt.



(b) CDFs for the three cases.



(c) PDFs for the three cases.

Fig. 3. Task completion latencies for a Microsoft Azure serverless function, for the different times between function invocations Δt: (a) Scatterplot of task completion latencies vs. Δt, and the CDFs (b) and the PDFs (c) for 3 distinct response time dynamics we identified in this data.

5.2. Latency changes with autoscaling and spin-down

Due to cloud providers internal resource allocation mechanisms that prioritize in-demand functions over functions accessed infrequently [60], serverless functions that are called more often execute faster. Industry professionals who work with serverless computing have been noting this [50]; however, we are unaware of attempts to capture these properties of serverless execution points for smart gateway-based decision-making.

To evaluate the behavior of serverless functions for different invocation rates, we conducted a set of experiments for which we varied the time between subsequent function invocations Δt via a timer we implemented on a laptop we used to represent a smart gateway. The differences between task completion times for functions invoked with different frequencies were substantial. For example, in our experiments with the PSF task in the North Virginia AWS datacenter, the difference between the median execution times of functions invoked within 1.5 s of each other and functions invoked with longer delays ranged from 7% to 42%.

When serverless functions are not invoked for longer periods of time, task completion latencies can increase further. For example, in our experiment with invoking an AWS function in a North Virginia datacenter every 30 min over the course of a day, the median response time was 0.95 s, for a function that responds in 0.5 s when called frequently — a *90% increase in latency*. In our experiment with invoking a Microsoft Azure function in the North Virginia datacenter every 30 min over the course of 1.5 days, the average response time was 6.0 s, for a function that normally responds in under 0.15 s — a *40x difference*. The differences we observe between these "cold start" times with different serverless platforms have been noted in [49].

5.3. Modeling latency in the presence of spin-down

To understand the potential approaches to modeling spin-down latency increases, we conducted a longer-term experiment with one particular execution option: 20,000 consecutive Microsoft Azure function invocations in North Virginia, with randomly chosen times between the invocations Δt, with the maximum $\Delta t = 5$ min (300 s). This experiment took more than 18 days. The experiment focused on the effects of spin-down, rather than on service scale-out. That is, we invoked the serverless function from only one location, thus the time between the invocations was never less than the minimum function response time of 0.06 s. The scatterplot of the obtained response times versus Δt is shown in Fig. 3(a).[2]

In this data, we were able to isolate three distinct cases: $\Delta t < 10$ s ($N = 10{,}425$), 10 s $< \Delta t < 60$ s ($N = 1641$), and $\Delta t > 60$ s ($N = 7877$), where N corresponds to the number of task latency samples falling in the identified range. The CDFs and the PDFs for these three cases are shown in Figs. 3(b) and 3(c). The task completion latencies for the cases of $\Delta t < 10$ s and $\Delta t > 60$ s are drawn from different distributions; the task completion distribution for the case of 10 s $< \Delta t < 60$ s has the statistical properties of the mixture of the other two distributions, as can be observed, for example, in Fig. 3(c).

5.4. Gateway-based decision-making with serverless execution options

From the point of view of task execution point selection in fog computing architectures, serverless autoscaling translates to having *execution points with near-infinite capacities*. Correspondingly, it is useful to include infinite-capacity execution points into fog task scheduling and resource allocation problem formulations.

Additionally, from the point of view of assigning tasks to execution points, serverless function autoscaling and spin-down lead to a paradoxical, counter-intuitive behavior: *the more work we assign to a serverless execution option, the faster it responds*. A gateway that is serving multiple IoT nodes would lower the response latency if it assigned multiple nodes' tasks to the same serverless execution point.

The underlying mechanisms controlling task completion latencies in serverless execution are complex; their details are not usually revealed to the users.[3] A gateway that attempts to empirically obtain a precise task latency characterization that will be used in task assignment algorithms runs into the following problems:

- The gateway needs to probe the serverless execution point *at the expected task invocation frequency*, as the observations made at other invocation frequencies may be substantially different.
- As an individual gateway does not have the information about the frequency of invocation requests generated by the other gateways, the gateway does not have the full knowledge of function inter-invocation times, and hence cannot provide accurate latency estimates.

Thus in many practical cases gateway-based predictions of serverless execution latencies could be closer to pessimistic estimates than to precise characterizations.

6. Task completion times as random variables

In this section we demonstrate how task completion times can be modeled as random variables (Section 6.1) and comment on the distributions of different execution options (Section 6.2).

[2] This plot excludes 64 outlier points with response times > 0.5 s.

[3] As observed in [61], by not revealing the details of the underlying infrastructure to the users, service providers can evolve the infrastructure without getting locked into outdated design decisions, and also avoid revealing trade secrets.

6.1. Fits of Generalized Extreme Value (G.E.V.) distributions to experimental latency values

To better understand the properties of task completion latencies, we fitted the experimentally obtained benchmark completion latency samples to a set of 20 common statistical distributions using the conventional maximum likelihood estimation (MLE) technique to estimate the parameter vectors for the hypothesized distributions. It is well-known that the MLE technique is *asymptotically optimal* under mild conditions since it achieves the Cramer–Rao lower bound and becomes an unbiased estimator for large datasets [62]. Due to its desirable asymptotical features, the MLE approach employed in this paper is also the common approach in many practical estimation problems in which the sufficient statistics cannot be obtained analytically for finding the minimum variance unbiased estimator. The full set of distributions we examined is as follows: Beta, Binomial, Birnbaum–Saunders, Exponential, Extreme Value, Gamma, Generalized Extreme Value, Generalized Pareto, Inverse Gaussian, Logistic, Log–logistic, Lognormal, Nakagami, Negative Binomial, Normal, Poisson, Rayleigh, Rician, *t* Location-scale, and Weibull.

Having obtained estimated distributions following the MLE approach, we order the quality of fitted distributions by analyzing the total deviation between the estimated probability distributions and the numerical one obtained using the experimental data. For this purpose, the usual metric employed is the total variation distance, which forms a metric space on the set of probability distribution functions [63].[4] The distribution that is best-fit to the data is defined as the one minimizing the distance to the numerical distribution function. The second-best fit distribution is defined similarly.

We observe that in many of the cases we examine, experimental latency values match well to *Generalized Extreme Value (G.E.V.)* distributions with a positive shape parameter, that is, a Type II G.E.V. distribution, which are also known as Frechet or Inverse Weibull.

For example, we closely examined the following three datasets we collected:

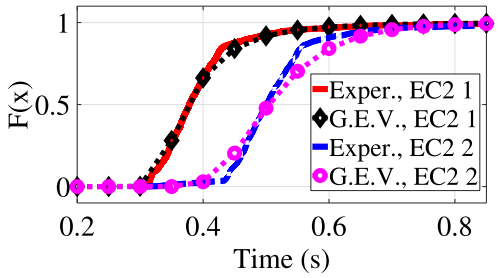
(i) A dataset with 34,449 samples of the PIC benchmark executed on execution points 1, 3 (both locations), and 4 (both locations), with execution options of 5000, 50,000, and 500,000 iterations. The executions were requested from an on-campus wirelessly connected laptop. The total duration of this data collection experiment was approximately 8 h.

(ii) A dataset with 41,693 samples of the PSF benchmark executed on execution points 1, 3 (both locations), and 4 (both locations), with execution options of 500, 5000, and 50,000 lines read. The executions were requested from an on-campus wirelessly connected laptop. The total duration of this data collection experiment was approximately 6 h.

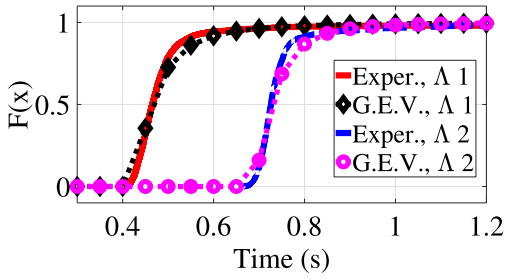
(iii) A dataset with 56,392 samples of the FSP benchmark executed on execution points 1, 2, and 3 (both locations), with execution options of 500, 5000, and 10,000 lines transmitted. The executions were requested from a wirelessly connected laptop in an off-campus residence. The total duration of this data collection experiment was approximately 11 h.

In this case, for the PIC benchmark (settings (i)), the G.E.V. distribution is the best fit in 9 out of 15 experiments, and second-best fit in 3 additional experiments. For the PSF benchmark (settings (ii)), the G.E.V. distribution is the best fit in 8 out of 15 experiments, and second-best fit in the additional 7 — that is, in every experiment a G.E.V. distribution is either the best or the second-best fit. For the FSP benchmark (settings (iii)), while the best fit is typically a *t* Location scale distribution (in 10 out of 12 experiments) or a logistic distribution (in the remaining 2 experiments), the G.E.V. distribution is a second-best fit in 7 out of 12 experiments.

[4] For two probability distributions μ and ν defined on a metric space X, the total variation distance $d_{\mathrm{TV}}(\mu,\nu)$ between them is given by $d_{\mathrm{TV}}(\mu,\nu) = \sup_{A\subseteq X} |\mu(A) - \nu(A)|$.



(a) AWS EC2.



(b) AWS Lambda ("Λ").

Fig. 4. The CDFs of measured latencies for several settings, shown with the best-fitting Generalized Extreme Value (G.E.V.) distributions. In many cases we examine, the G.E.V. is the best or the second-best fit out of 20 common statistical distributions.

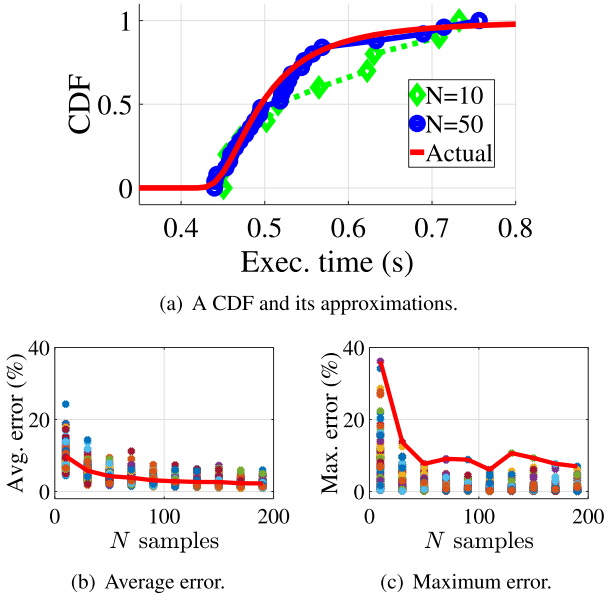
Table 8
G.E.V. distribution fits to the experimental latency data, for some of the examined benchmarks.

Settings	*N*	Distribution support (s)	G.E.V. parameters
PIC, settings (i), 500,000 iter., execution point 1	1444	0.42–5.8	Shape 0.34, scale 0.04, location 0.48
PSF, settings (ii), 5000 lines, execution point 3, US West datacenter	3961	0.15–3.2	Shape 0.35, scale 0.004, location 0.16
FSP, settings (iii), 10,000 lines, execution point 1	2710	0.09–0.64	Shape 0.12, scale 0.008, location 0.11

Table 8 shows the parameters of the fitted G.E.V. distributions for three of the experiments with *N* task completion latency samples, for which the G.E.V. is the best-fitting or the second-best-fitting distribution of all examined. In these experiments, the means of the fitted distributions, which are identified by the *location* parameter in G.E.V. distributions, are substantially different; the extent to which the distributions are spread out, identified by the *scale* parameter, is different as well. In all cases, the G.E.V. *shape* parameter is positive, corresponding to the Inverse Weibull distributions. Fig. 4 shows several experimentally obtained CDFs and the CDFs of fitted G.E.V. distributions.

Inverse Weibull distributions have previously been observed to fit HTTP server processing times [64]; as the underlying communication mechanism in our experiments is HTTP, it is logical that these distributions would be a fit for some experiments, especially those where HTTP-related mechanisms dominate task completion latencies.

We also observe a close fit of G.E.V. distributions to serverless execution points' latencies. For example, for the serverless execution options in the above-described experiments, a G.E.V. distribution is the best fit in *11 out of 12 experiments*. Examples of a fit of a G.E.V. distribution to serverless execution points' latencies are shown in Fig. 4(b).

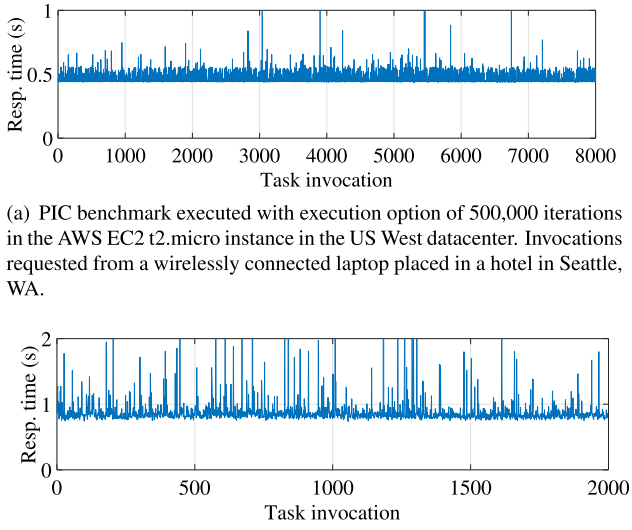


(a) A CDF and its approximations.

(b) Average error.

(c) Maximum error.

Fig. 5. An experimentally obtained execution point latency CDF F and its estimates $\hat{F}_N$ based on $N = 10$ and $N = 50$ samples from the distribution (a), and the average (b) and the maximum (c) distance between the experimental F and its estimates $\hat{F}_N$, for the different number of samples N. Even a small number of samples N allows characterizing a CDF well.



(a) PIC benchmark executed with execution option of 500,000 iterations in the AWS EC2 t2.micro instance in the US West datacenter. Invocations requested from a wirelessly connected laptop placed in a hotel in Seattle, WA.

(b) FSP benchmark executed with execution option of 50,000 lines read on the AWS Lambda serverless platform in the US West datacenter. Invocations requested from an on-campus wirelessly connected laptop.

Fig. 6. Response time for multiple task invocations, for two different settings. Response time distribution remains largely stable throughout the multi-hour duration of the experiments.

G.E.V. distributions typically arise in the study of order statistics. We hypothesize that internal proprietary cloud providers' service provisioning mechanisms behind serverless operations involve a minimization or a maximization of a performance quality, and thus are readily described by the G.E.V. distributions.

Our observations about the fits of G.E.V. distributions to task latencies are limited to low-complexity tasks we use in our benchmarking. Complex operations with runtimes that vary widely for different inputs (e.g., cascading classifiers with multiple exit points [65], or complex interactive applications with many components [14]), may have task completion latency distributions defined by algorithm-related, rather than infrastructure-related, phenomena. We do, however, expect low-complexity tasks to represent an important fraction of the IoT tasks served by fog architectures.

6.2. Distributions for different execution options

In addition to examining the fits, we also considered whether the distributions are similar for the different execution points and execution options. We found that the distributions can vary. For instance, for the PIC benchmark, we saw that the increase in the complexity of a task led to a substantial increase in not just the median latency, but also in the span of the latency distribution (i.e., the shape of the CDF is different for different execution options). For the FSP benchmark, with the increase in the complexity of the task, the span of the distribution increased slightly for execution points 2 and 3, and substantially for execution point 1. Overall, the distributions can be very different for different execution options. Thus, the characteristics of the latency of different execution options need to be obtained separately; we cannot assume that the CDFs will be the "shifted versions" of each other.

7. Estimating task completion times based on partial information

In this section we provide insights into characterizing task completion latency CDFs based on a small numbers of task completion latency samples.

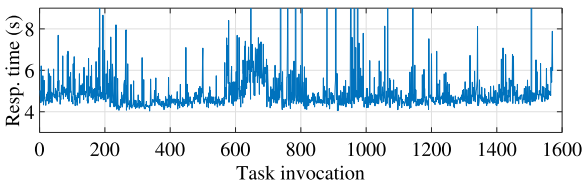
The differences in CDFs of different execution options (see above) suggest that the characterizations of the different options need to be separately obtained. For multi-point multi-quality fog systems, the need to separately characterize each execution option on each execution point may seem daunting. Fortunately, a gateway would require few latency samples to obtain an approximation of a task completion latency CDF, as we show below.

Fig. 5(a) shows an experimentally obtained CDF F for setting 1 in Table 8, in comparison with Kaplan–Meier estimates of this CDF, $\hat{F}_N$, based on 10 and 50 samples N from this distribution. It can be observed that for these values of N, F and $\hat{F}_N$ are close to each other. To quantify the proximity, Fig. 5(b, c) show the average and the maximum distances between F and $\hat{F}_N$ for different N values for this distribution. For each value of N, samples were randomly drawn 100 times; the dots on the graphs show the individual results, while the line shows the results' average in Fig. 5(b), and the results' maximum in Fig. 5(c). It can be seen, for example, that when 30 task completion latency samples are used to obtain the CDF estimate $\hat{F}_N$, the average distance between the estimated and the empirical CDF is only 6%, and the maximum error is only 17%.

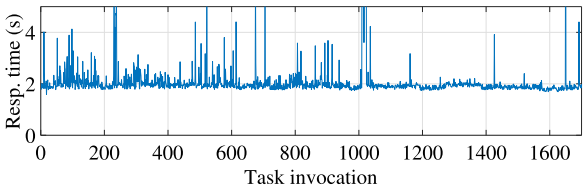
Obtaining task completion latency samples to characterize execution point latencies should not be onerous in practice, as it is straightforward for the gateway to measure latencies while executing tasks. That is, in the example above, when the gateway executes a task every minute, it would take 30 min for the estimated CDF to be within 6% of the empirical CDF on average. Approaches that trade off exploitation and exploration in latency characterization can be developed to further reduce the execution option characterization overhead.

8. Task completion latency stability

In this section we examine the stability of task completion latency distributions with respect to time. We first comment on task completion latency distribution stability we observed in a range of experiments that lasted on the order of hours (Section 8.1), and then describe the distribution stability we observed in two multi-day experiments (Section 8.2).

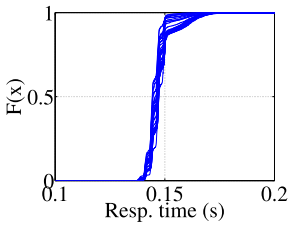


(a) PIC benchmark with execution option of 500,000 iterations. Invocations requested from a laptop connected to in-flight WiFi on a flight from Europe to the US, and executed on the AWS Lambda platform US East datacenter.

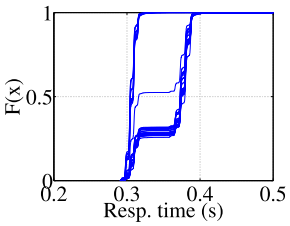


(b) PIC benchmark with execution option of 500,000 iterations. Invocations requested from a laptop connected to in-flight WiFi on a flight from Houston to Seattle, and executed in the AWS EC2 platform US East datacenter.

Fig. 7. Response time for multiple task invocations, for two different in-flight experiments. While the distributions vary more than the distributions in less challenged conditions, they also exhibit long-term stability.



(a) FSP, lower network load.



(b) FSP, higher network load.

Fig. 8. The overlayed CDFs of 5000 consecutive task executions intervals in a 10-day experiment, for an AWS EC2 instance-based execution point, for the FSP benchmark with lower (a) and higher (b) network utilization. Over the 10-day experiment, latency characteristics of the execution point remained stable for one experiment (a), and changed once for the other (b).

8.1. Latency stability in multi-hour experiments

In our experiments we observed that the statistics of task completion latencies in a given environment remained largely the same for our multi-hour experiments. Two representative cases are shown in Fig. 6, which demonstrates latency measurements as a function of time for 2 different settings: (a) 8000 executions of the PIC benchmark, with the execution option of 500,000 iterations, called from a laptop placed in a hotel in Seattle, WA, and using the hotel WiFi, executed in an AWS EC2 t2.micro instance in the US West datacenter, and (b) 2000 executions of the PSF benchmark, with the execution option of 50,000 lines read, called from a laptop placed on campus and using a campus WiFi, executed on the serverless AWS Lambda platform in the US West datacenter. Fig. 6 shows that, while the response times are random, both the average response time and the level of variation around the average are stable with respect to time in both settings.

To quantify the stability of response time distributions, we consider the stability of *distribution percentiles*. Specifically, taking consecutive 300-sample intervals of the recorded response times and obtaining their statistics, we compare the medians (50th percentiles) and the 90th and the 95th percentiles of the 300-sample distributions. For the settings described above and shown in Fig. 6, these percentiles are close to each other. For instance, for the PIC task (Fig. 6(a)), the highest difference between the distribution medians is 0.7%, and the highest difference between the 95th percentiles is 6%. For the FSP benchmark (Fig. 6(b)), the highest difference between the distribution medians is 2.8%, and the highest difference between the 95th percentiles is 26%. In the numerous multi-hour experiments we conducted, such stability of a CDF with respect to time is typical, observed in nearly all cases. The stability indicates that *an obtained task completion latency CDF is likely to remain useful for long periods of time*.

The only set of conditions where we observed some variability are the in-flight WiFi conditions. Some of our in-flight WiFi experiments are described in Section 4 (experiments 4, 6, and 10 in Table 7, and the distributions demonstrated in Fig. 2); we also conducted several other in-flight experiments, where, similarly to the experiments described in Section 4, the median latencies were significantly larger than latencies in non-flight environments. However, even in those challenged conditions, despite a dramatic increase in the overall magnitude of latency compared to other conditions, the latency distribution is more stable than one may expect. Two examples are shown in Fig. 7. They correspond to the following conditions: (a) 1600 executions of the PIC benchmark with the execution option of 500,000 iterations, called from a laptop connected to in-flight WiFi on a flight from Europe to the US, and executed on the serverless AWS Lambda platform in the US East datacenter, and (b) 1700 executions of the PIC benchmark with the execution option of 500,000 iterations, called from a laptop connected to in-flight WiFi on a flight from Houston, TX to Seattle, WA, and executed on the AWS EC2 platform in the US East datacenter. Here, in the first set of conditions, the highest difference between the medians is 7.2%, and the highest difference between the 95th percentiles is 30%. In the second set of conditions, the highest difference between the medians is 6.7%, and the highest difference between the 95th percentiles is 83%. While significantly more varied than in non-challenged conditions, the distributions are still stable enough for their characterizations to be useful over long periods of time.

8.2. Latency stability in multi-day experiments

To examine latency CDF stability further, we conducted a 10-day experiment with the FSP benchmark executing in an AWS EC2 instance in a US West datacenter (>800,000 consecutive executions), and a 2-day experiment with the PIC benchmark executing in an AWS EC2 instance in a US East datacenter (>200,000 consecutive executions). To exclude WiFi-related disruptions, we conducted these experiments with a laptop connected via Ethernet.

Our results demonstrate that task completion latencies' properties remain stable for long periods of time, undergoing changes only once in a while. For example, for the FSP benchmark executed with the option of a small dataset transmitted over the network (500 lines transmitted), the statistical properties of task completion latencies *remained stable over the entire 10-day interval*. Fig. 8(a) shows, for this case, the CDFs for the successive 5000-invocation intervals of this experiment, overlayed with each other. Throughout the 10 days of this experiment, the 5th, 10th, 25th, 50th, and 75th percentiles of task completion latencies stayed within 2.2% from each other, the 90th and the 95th percentiles — within 8.5%. For the FSP benchmark executed with the option of a larger dataset transmitted over the network (10,000 lines transmitted), *over the 10-day course of the experiment, the CDF statistics changed once*, on the 4th day of the experiment. Fig. 8(b) shows, for this case, the CDFs for the successive 5000-invocation intervals of this benchmark. The difference between the percentile values for the different CDFs in this case is up to 20%; for the CDFs before and after the 4th-day transition the difference in all percentiles does not exceed 2.1%.

For the PIC benchmark, the CDF statistics changed only once as well, in a 2-day interval, but the changes were more pronounced: for one of the options, the median task completion latency increased

6x (from 0.05 s to 0.32 s), and for the other — 10x (from 0.29 s to 2.9 s). We hypothesize that this notable change is related to CPU sharing in the AWS t2.micro instance we use, as the PIC is a CPU-sensitive benchmark. Similarly to the case of the FSP, before and after the transition, task completion latency CDFs remained stable. Over the 17 h before the transition and the 27 h after the transition, the CDFs' percentiles differed by no more than a few percent.

The observed stability in task completion latencies indicates that the execution point latency characterizations can be conducted infrequently, but do need to be updated periodically. In particular, the extent of task completion latency changes we observed for the PIC benchmark – e.g., from 0.29 s to 2.9 s – is *on the order that would call for selecting a different task execution point for a latency-sensitive task*.

9. Conclusions

Fog computing is receiving increasing attention from industry and academia alike due in part to its potential for enabling advances in the Internet of Things (IoT) applications. Yet, few comprehensive quantitative characterizations of the properties of fog computing architectures have been conducted. In this work we examined statistical properties of task completion latencies in fog computing systems with multiple heterogeneous execution points. The experimental study we conducted covered a range of settings, and uniquely considered both traditional and serverless fog computing execution points. Our study elucidated several important properties of task execution latencies, including the heterogeneity of their properties across different locations and the need for separate characterizations of different execution options. Based on the results of our study, we demonstrated how task completion latencies can be modeled as random variables. We also demonstrated that such models can be obtained from a limited number of task completion latency samples, and that the obtained statistical characterizations are likely to remain useful over long periods of time.

In future work we will expand our benchmarks to include popular cloud-based services for IoT devices, such as AWS Rekognition [66], a service that processes device-provided images to extract labels of objects detected in the images. We will also design fog task allocation algorithms that take the execution points' and options' properties we observed in this study into account.

CRediT authorship contribution statement

Maria Gorlatova: Conceptualization, Methodology, Data curation, Writing - original draft, Writing - review & editing. **Hazer Inaltekin:** Conceptualization, Formal analysis, Writing - review & editing. **Mung Chiang:** Supervision, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by the Comcast Innovation Fund Research Grant, AWS Cloud Credits for Research, Microsoft Azure Research Award, National Science Foundation (NSF) grant CSR-1812797, Australian Research Council grant DP-20-0101627, and Defense Advanced Research Projects Agency (DARPA) under contract No. HR0011-17C0052 and No. HR001117C0048. The opinions, findings and conclusions expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency.

References

[1] OpenFog reference architecture, 2017, www.openfogconsortium.org/ra/.
[2] M. Chiang, T. Zhang, Fog and IoT: An overview of research opportunities, IEEE Internet Things J. 3 (6) (2016) 854–864.
[3] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the Internet of Things, in: Proc. ACM MCC'12, 2012.
[4] ETSI Industry Specification Group, MEC Framework and reference architecture, 2017, www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing.
[5] Amazon Inc., AWS Greengrass, https://aws.amazon.com/greengrass.
[6] Amazon Inc., Lambda at edge, https://docs.aws.amazon.com/lambda/latest/dg/lambda-edge.html.
[7] Microsoft Inc., Azure IoT edge, https://github.com/Azure/iot-edge.
[8] H. Tan, Z. Han, X.-Y. Li, F.C. Lau, Online job dispatching and scheduling in edge-clouds, in: Proc. IEEE INFOCOM'17, 2017.
[9] M. Jia, J. Cao, W. Liang, Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks, IEEE Trans. Cloud Comput. 5 (4) (2017) 725–737.
[10] S. Luo, X. Chen, Z. Zhou, F3C: Fog-enabled joint computation, communication and caching resource sharing for energy-efficient IoT data stream processing, in: Proc. IEEE ICDCS'19, 2019.
[11] W. Zhang, S. Li, L. Liu, Z. Jia, Y. Zhang, D. Raychaudhuri, Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds, in: Proc. IEEE INFOCOM'19, 2019.
[12] P. Kortoçi, L. Zheng, C. Joe-wong, M.D. Francesco, M. Chiang, Fog-based data offloading in urban IoT scenarios, in: Proc. IEEE INFOCOM'19, 2019.
[13] G. Premsankar, M. Di Francesco, T. Taleb, Edge computing for the Internet of Things: A case study, IEEE Internet Things J. 5 (2) (2018) 1275–1284.
[14] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. Klatzky, D. Siewiorek, M. Satyanarayanan, An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance, in: Proc. ACM SEC'17, 2017.
[15] V.B. Souza, X. Masip-Bruin, E. Marin-Tordera, W. Ramirez, S. Sanchez, Towards distributed service allocation in fog-to-cloud (f2c) scenarios, in: Proc. IEEE GLOBECOM'16, 2016.
[16] Y. Xiao, M. Krunz, QoE and power efficiency tradeoff for fog computing networks with fog node cooperation, in: Proc. IEEE INFOCOM'17, 2017.
[17] X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing, IEEE/ACM Trans. Netw. 24 (5) (2016) 2795–2808, http://dx.doi.org/10.1109/TNET.2015.2487344.
[18] C. You, K. Huang, H. Chae, B. Kim, Energy-efficient resource allocation for mobile-edge computation offloading, IEEE Trans. Wireless Commun. 16 (3) (2017) 1397–1411, http://dx.doi.org/10.1109/TWC.2016.2633522.
[19] C. Zhu, G. Pastor, Y. Xiao, Y. Li, A. Ylae-Jaeaeski, Fog following me: Latency and quality balanced task allocation in vehicular fog computing, in: Proc. IEEE SECON'18, 2018, http://dx.doi.org/10.1109/SAHCN.2018.8397129.
[20] L. Wang, L. Jiao, T. He, J. Li, M. Mühlhäuser, Service entity placement for social virtual reality applications in edge computing, in: Proc. IEEE INFOCOM'18, 2018.
[21] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, A. Krishnamurthy, MCDNN: An approximation-based execution framework for deep stream processing under resource constraints, in: Proc. ACM MobiSys'16, 2016.
[22] X. Ran, H. Chen, X. Zhu, Z. Liu, J. Chen, DeepDecision: A mobile deep learning framework for edge video analytics, in: Proc. IEEE INFOCOM'18, 2018.
[23] A.P. Iyer, L.E. Li, M. Chowdhury, I. Stoica, Mitigating the latency-accuracy trade-off in mobile data analytics systems, in: Proc. ACM MobiCom'18, 2018.
[24] Y. Li, Y. Chen, T. Lan, G. Venkataramani, MobiQoR: Pushing the envelope of mobile edge computing via quality-of-result optimization, in: Proc. ICDCS'17, 2017.
[25] Z. Liu, G. Lan, J. Stojkovic, Y. Zhang, C. Joe-Wong, M. Gorlatova, CollabAR: Edge-assisted collaborative image recognition for augmented reality, in: Proc. ACM/IEEE IPSN'20, 2020.
[26] P. Georgiev, N.D. Lane, K.K. Rachuri, C. Mascolo, LEO: Scheduling sensor inference algorithms across heterogeneous mobile processors and network resources, in: Proc. ACM MobiCom'16, 2016.
[27] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, R. Govindan, Odessa: Enabling interactive perception applications on mobile devices, in: Proc. ACM MobiSys'11, 2011.
[28] H. Inaltekin, M. Gorlatova, M. Chiang, Virtualized control over fog: Interplay between reliability and latency, IEEE Internet Things J. 5 (6) (2018) 5030–5045.
[29] Z. Qi, P. Dong, K. Ma, N. Sargeant, A design of in-car multi-layer communication network with Bluetooth and CAN bus, in: Proc. IEEE AMC'16, 2016.
[30] V. Liberatore, Integrated play-back, sensing, and networked control, in: Proc. IEEE INFOCOM'06, 2006.
[31] Dell PowerEdge R930 rack server, www.dell.com/en-us/work/shop/povw/poweredge-r930.
[32] Amazon Inc., AWS EC2 instance types, https://aws.amazon.com/ec2/instance-types/.
[33] Amazon Inc., AWS Lambda, https://aws.amazon.com/lambda/.

[34] C. Streiffer, A. Srivastava, V. Orlikowski, Y. Velasco, V. Martin, N. Raval, A. Machanavajjhala, L.P. Cox, ePrivateEye: To the edge and beyond!, in: Proc. ACM SEC'17, 2017.
[35] Google, Cloud functions, https://cloud.google.com/functions/.
[36] Microsoft Inc., Azure functions, https://azure.microsoft.com/en-us/services/functions/.
[37] IBM Inc., Bluemix OpenWhisk, www.ibm.com/cloud-computing/bluemix/openwhisk.
[38] M. Gorlatova, H. Inaltekin, M. Chiang, Dataset: Fog latency characterization, 2019, https://maria.gorlatova.com/public-datasets/fog-latency-characterization/.
[39] J. Du, L. Zhao, J. Feng, X. Chu, Computation offloading and resource allocation in mixed fog/cloud computing systems with min–max fairness guarantee, IEEE Trans. Commun. 66 (4) (2018) 1594–1608, http://dx.doi.org/10.1109/TCOMM.2017.2787700.
[40] L. Yin, J. Luo, H. Luo, Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing, IEEE Trans. Ind. Inf. 14 (10) (2018) 4712–4721, http://dx.doi.org/10.1109/TII.2018.2851241.
[41] Z. Zhou, P. Liu, J. Feng, Y. Zhang, S. Mumtaz, J. Rodriguez, Computation resource allocation and task assignment optimization in vehicular fog computing: A contract-matching approach, IEEE Trans. Veh. Technol. 68 (4) (2019) 3113–3125, http://dx.doi.org/10.1109/TVT.2019.2894851.
[42] X. Chen, J. Zhang, When D2D meets cloud: Hybrid mobile task offloading in fog computing, in: Proc. IEEE ICC'17, 2017, pp. 1–6, http://dx.doi.org/10.1109/ICC.2017.7996590.
[43] Z. Maamar, T. Baker, N. Faci, E. Ugljanin, M.A. Khafajiy, V. Burégio, Towards a seamless coordination of cloud and fog: Illustration through the Internet-of-Things, in: Proc. ACM/SIGAPP Symposium on Applied Computing, 2019.
[44] K. Sui, M. Zhou, D. Liu, M. Ma, D. Pei, Y. Zhao, Z. Li, T. Moscibroda, Characterizing and improving WiFi latency in large-scale operational networks, in: Proc. ACM MobiSys'16, 2016.
[45] Z. Wu, H.V. Madhyastha, Understanding the latency benefits of multi-cloud webservice deployments, SIGCOMM Comput. Commun. Rev. 43 (2) (2013) 13–20, http://dx.doi.org/10.1145/2479957.2479960, URL http://doi.acm.org/10.1145/2479957.2479960.
[46] A. Li, X. Yang, S. Kandula, M. Zhang, CloudCmp: Comparing public cloud providers, in: Proc. ACM IMC'10, 2010.
[47] Y. Gao, W. Hu, K. Ha, B. Amos, P. Pillai, M. Satyanarayanan, Are Cloudlets Necessary? Tech. Rep. CMU-CS-15-139, CMU School of Computer Science, 2015.
[48] C.A. García-Pérez, P. Merino, Experimental evaluation of fog computing techniques to reduce latency in LTE networks, Trans. Emerg. Telecommun. Technol. 29 (4) (2017).
[49] G. McGrath, P.R. Brenner, Serverless computing: Design, implementation, and performance, in: Proc. IEEE ICDCSW'17, 2017, http://dx.doi.org/10.1109/ICDCSW.2017.36.
[50] G. Neves, Keeping functions warm - how to fix AWS Lambda cold start issues, Serverless Blog, 2017, https://www.serverless.com/blog/keep-your-lambdas-warm/.
[51] S.K. Mohanty, G. Premsankar, M. di Francesco, An evaluation of open source serverless computing frameworks, in: Proc. IEEE CloudCom'18, 2018, pp. 115–120, http://dx.doi.org/10.1109/CloudCom2018.2018.00033.
[52] Z. Zheng, Y. Zhang, M.R. Lyu, Investigating QoS of real-world web services, IEEE Trans. Serv. Comput. 7 (1) (2014) 32–39, http://dx.doi.org/10.1109/TSC.2012.34.
[53] S. Choy, B. Wong, G. Simon, C. Rosenberg, The brewing storm in cloud gaming: A measurement study on cloud to end-user latency, in: Proc. ACM NetGames'12, 2012.
[54] K.-T. Chen, Y.-C. Chang, P.-H. Tseng, C.-Y. Huang, C.-L. Lei, Measuring the latency of cloud gaming systems, in: Proc. ACM MM'11, 2011.
[55] Super Pi benchmark, www.superpi.net.
[56] A. Ronacher, Flask: Web development, one drop at a time, https://flask.pocoo.org/.
[57] RouteHappy, RouteHappy 2018 Wi-Fi Report Evaluates Global In-Flight Wi-Fi, 2018, https://www.routehappy.com/insights/wi-fi/2018/.
[58] A. Glikson, S. Nastic, S. Dustdar, Deviceless edge computing: Extending serverless computing to the edge of the network, in: Proc. ACM SYSTOR'17, 2017.
[59] E. van Eyk, L. Toader, S. Talluri, L. Versluis, A. Uta, A. Iosup, Serverless is more: From PaaS to present cloud computing, IEEE Internet Comput. 22 (5) (2018) 8–17, http://dx.doi.org/10.1109/MIC.2018.053681358.
[60] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, P. Suter, Serverless computing: Current trends and open problems, in: S. Chaudhary, G. Somani, R. Buyya (Eds.), Research Advances in Cloud Computing, Springer Singapore, 2017.
[61] J.C. Mogul, R.R. Kompella, Inferring the network latency requirements of cloud tenants, in: Proc. ACM HotOS'15, 2015.
[62] H.V. Poor, An Introduction to Signal Detection and Estimation, second ed., Springer-Verlag, New York, NY, 1994.
[63] R. Durrett, Probability: Theory and Examples, second ed., Duxbury Press, Belmont, CA, USA, 1996.
[64] J. Cao, W.S. Cleveland, Y. Gao, K. Jeffay, F.D. Smith, M. Weigle, Stochastic models for generating synthetic HTTP source traffic, in: Proc. IEEE INFOCOM'04, 2004.
[65] S. Teerapittayanon, B. McDanel, H.T. Kung, Distributed deep neural networks over the cloud, the edge and end devices, in: Proc. IEEE ICDCS'17, 2017.
[66] Amazon Inc., AWS Rekognition, https://aws.amazon.com/rekognition/.

Maria Gorlatova received the B.Sc. (summa cum laude) and M.Sc. degrees in electrical engineering from the University of Ottawa, Ottawa, ON, Canada, and the Ph.D. degree in electrical engineering from Columbia University, New York, NY, USA. She is a Nortel Networks Assistant Professor of Electrical and Computer Engineering and Computer Science at Duke University, Durham, NC, USA. She has several years of industry experience, and has been affiliated with Telcordia Technologies, Piscataway, NJ, USA, IBM, Armonk, NY, USA, and D. E. Shaw Research, New York, NY, USA. Her current research interests include architectures, algorithms, and protocols for emerging pervasive technologies. Dr. Gorlatova was a recipient of the Google Anita Borg USA Fellowship, the Canadian Graduate Scholar NSERC Fellowships, and the Columbia University Presidential Fellowship. She was a co-recipient of the 2011 ACM SenSys Best Student Demonstration Award, the 2011 IEEE Communications Society Award for Advances in Communications, the 2020 IEEE/ACM IPSN Best Research Artifact Award, and the 2016 IEEE Communications Society Young Author Best Paper Award.

Hazer Inaltekin received the B.S. degree (High Hons.) in electrical and electronics engineering from Bogaziçi University, Istanbul, Turkey, in 2001, and the M.S. and Ph.D. degrees in electrical and computer engineering from Cornell University, Ithaca, NY, USA, in 2006. He held various researcher and faculty positions in Australia, Europe, and the U.S. He is a Senior Lecturer at the Macquarie University, Sydney, Australia. His current research interests include fog computing, IoT technology, wireless communications, wireless networks, social networks, game theory, and information theory.

Mung Chiang was the Arthur LeGrand Doty Professor of Electrical Engineering with Princeton University, Princeton, NJ, USA, where he also served as the Director of Keller Center for Innovations in Engineering Education and the inaugural Chairman of Princeton Entrepreneurship Council. He is the John A. Edwardson Dean of the College of Engineering and the Roscoe H. George Professor of Electrical and Computer Engineering with Purdue University, West Lafayette, IN, USA. His textbook Networked Life, popular science book The Power of Networks, and online courses reached over 400 000 students since 2012. He founded the Princeton EDGE Lab in 2009, which bridges the theory-practice gap in edge networking research by spanning from proofs to prototypes. He also co-founded a few startup companies in mobile data, IoT, and AI, and co-founded the global nonprofit Open Fog Consortium. Mr. Chiang was a recipient of the 2013 Alan T. Waterman Award for his research on networking, the highest honor to U.S. young scientists and engineers.