The Hidden Cost of the Edge: A Performance Comparison of Edge and Cloud Latencies

Ahmed Ali-Eldin
Chalmers University of Technology
ahmed.hassan@chalmers.se

Bin Wang
University of Massachusetts Amherst
binwang@cs.umass.edu

Prashant Shenoy
University of Massachusetts Amherst
shenoy@cs.umass.edu

ABSTRACT

Edge computing has emerged as a popular paradigm for running latency-sensitive applications due to its ability to offer lower network latencies to end-users. In this paper, we argue that despite its lower network latency, the resource-constrained nature of the edge can result in higher end-to-end latency, especially at higher utilizations, when compared to cloud data centers. We study this edge performance inversion problem through an analytic comparison of edge and cloud latencies and analyze conditions under which the edge can yield worse performance than the cloud. To verify our analytic results, we conduct a detailed experimental comparison of the edge and the cloud latencies using a realistic application and real cloud workloads. Both our analytical and experimental results show that even at moderate utilizations, the edge queuing delays can offset the benefits of lower network latencies, and even result in performance inversion where running in the cloud would provide superior latencies. We finally discuss practical implications of our results and provide insights into how application designers and service providers should design edge applications and systems to avoid these pitfalls.

ACM Reference Format:
Ahmed Ali-Eldin, Bin Wang, and Prashant Shenoy. 2021. The Hidden Cost of the Edge: A Performance Comparison of Edge and Cloud Latencies . In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21), November 14–19, 2021, St. Louis, MO, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3458817.3476142

1 INTRODUCTION

Over the past decade, cloud computing has emerged as a popular paradigm for running a variety of distributed systems applications ranging from web applications to Al and high performance computing workloads. In recent years, edge computing has emerged as a complement to cloud computing, particularly for running latency-sensitive workloads. Edge computing involves using computational and storage resources deployed at the edge of the network to run applications that can benefit from low network latency. Emerging edge applications include mobile augmented reality, Internet of Things (IoT) analytics, and edge Al involving real-time inference over machine learning models. With the emergence of these edge workloads, many cloud providers have begun to offer edge cloud services by deploying edge servers clusters close to end users (akin to cloudlets [27]) and offering cloud-like service from the edge.

Since edge resources are deployed at the edge of the network, conventional wisdom holds that the edge is "better" than the cloud from a latency perspective, and thus well suited for any workload that has tight latency requirements. In this paper, we show that this conventional wisdom does not always hold. In particular, edge applications typically require low *end-to-end* latency, which consists of both the network and the server latency. While the edge provides a significantly lower network latency than the cloud, edge resources tend to be more constrained than those in cloud data centers. Consequently, workload dynamics can cause edge server latency to exceed the cloud server latency due to higher queuing delays at edge sites. In such cases, the total end-to end latency of the edge can *exceed* the end-to-end cloud latency, since the lower network latency of the edge is offset by a higher server latency at higher utilization levels. This "hidden" cost of the edge has not been studied previously. Our paper quantifies this cost through analytic and experimental comparisons of edge and cloud performance. Specifically, we use analytic queuing results and real world experimentation to characterize scenarios under which such performance inversion occurs.

In doing so, our performance comparison study makes the following contributions.

(1) We introduce and formulate the *edge performance inversion* problem, which causes the end-to-end latency of the edge to become higher than that of the cloud. We conduct an analytic performance comparison by using queuing models to to quantify the end-to-end latency of the edge and the cloud under different workload scenarios. We present closed-form analytic equations that specify conditions under which the total edge latency can exceed the cloud latency, causing a performance inversion.
(2) We conduct an experimental performance comparison of the edge and cloud performance using realistic applications and Azure trace workloads on real cloud and edge platforms. Our experiments show that for geo-distributed applications the mean and tail latencies of the edge can indeed be worse than the cloud even at moderate utilization levels of 40 to 60% and that such performance inversion is more likely in the presence of workload skews or as the latency to traditional clouds reduces due to increasing cloud deployments.
(3) Since the potential for performance inversion at the edge has important consequences for both application designers and cloud providers, we discuss the key implications for our results in practice. Specifically, we discuss resource allocation techniques and extra capacity needed at the edge to

SC '21, November 14–19, 2021, St. Louis, MO, USA

ACM ISBN 978-1-4503-8442-1/21/11…$15.00
https://doi.org/10.1145/3458817.3476142

avoid performance inversion with the cloud under different scenarios.

2 BACKGROUND

This section presents background on cloud and edge computing discussing the motivation for the edge performance inversion problem.

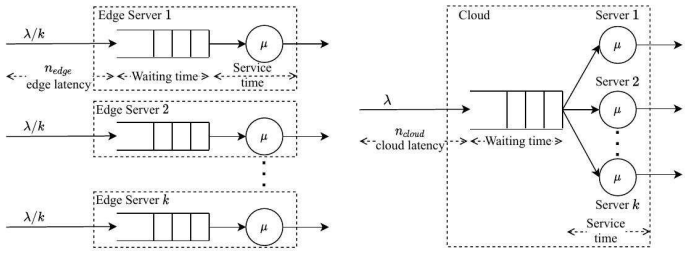
2.1 Cloud and Edge Computing

Traditional cloud computing involves deploying large-scale data centers that house tens of thousands of servers to run remote third-party applications. Customers can request these servers to run their applications and pay based on their resource usage on a pay-as-you-go basis. Cloud data centers are typically virtualized and allocate server resources to applications in the form of virtual machines. Today, cloud platforms are popular for running diverse applications ranging from online web services to scientific computing.

Edge computing and edge clouds are a natural evolution of traditional cloud computing [25, 34].They involve deploying server resources at the edge of the network to host workloads that are latency-sensitive in nature. While there are many forms of edge computing, in this work, we focus on edge clouds that involve deploying small server clusters, also known as micro data centers, at the geographically distributed sites to offer a cloud-like service from the network edge. The notion of edge clouds was originally proposed as an extension to the work on pervasive and mobile computing in the form of *Cloudlets*, which were nearby cluster that mobile devices could use to offload of their computations at low latency [26, 27]. Major cloud providers are now beginning to offer edge cloud services (e.g., Google's Stadia [10] and Azure IoT Edge service [1]). Edge clouds are especially useful for applications that demand low-latency access to cloud resources which can not be satisfied using more distant traditional cloud servers.

The benefits of the lower network latency offered by edge computing are well known, with recent studies showing the advantages of offloading to a local edge versus a remote cloud [11]. Others have taken a modeling approach to analyzing cloud and edge computing [4–6, 14, 15]. This has led to recent work on analytically studying edge clouds, e.g., to decide where and when services should be migrated in response to user mobility and demand variation [33], analytical models to compare the performance and utilization between single level and hierarchical designs of the edge clouds [30], and models to capture the energy consumption trade-offs when offloading the computations or running them locally [21].

Cloud workloads exhibit significant temporal and spatial dynamics [13, 20], and we expect edge workloads to exhibit a similar behavior. Temporal workload dynamics include diurnal time-of-day and seasonal effects as well as short-term burstiness and workload spikes in the form of flash crowds. Since the end-users accessing a cloud or edge application may be geographically distributed, the workload also exhibits spatial dynamics. If users are more concentrated at certain locations than others, the incoming workload will exhibit a skew in its spatial distribution. Moreover if the popularity of the application decreases in certain regions and rises in others, the spatial workload distribution will change over time. From the perspective of a traditional cloud applications, all requests arrive at a centralized data center and these temporal and spatial dynamics are handled at a single location via techniques such as dynamic resource allocation or elastic scaling [37]. However, in the case of a distributed edge cloud, different edge sites will see different spatial and temporal dynamics since an edge application such as an online game may be distributed across geographic sites and thus the overall workload will be partitioned across sites. We assume that applications have tight end-to-end latency requirements that need be satisfied by the remote server, whether at the edge or the cloud.



(a) Edge queuing model. (b) Cloud queuing model.

Figure 1: End-to-end latency of an edge and cloud deployment of an application.

2.2 Edge Performance Inversion: Motivation

Next, we motivate the potential for edge performance inversion using several intuitive observations. Consider an application that can either be deployed in the cloud or at the edge. Suppose that the round-trip network latency to the cloud server is n_{cloud} and that to the edge server is n_{edge}. Since edge resources are closer to end-users than cloud resources, we have $n_{edge} < n_{cloud}$. This lower network latency of edge servers has been the primary reason as to why edge computing has been assumed to be better than cloud computing for latency-sensitive applications.

From an application perspective, however, the performance is governed not just by network latency but by the total end-to-end latency. Let r_{edge} denote the response time of the edge server and r_{cloud} denote the response time of the cloud server for each application request. Thus, the end-to-end latency offered by the edge is $(n_{edge} + r_{edge})$ while that of the cloud is $(n_{cloud} + r_{cloud})$. As shown in Figure 1, the server response time itself consists two components; queuing delays at the server and the request execution time.

Bank teller analogy: Suppose that the application runs on a cluster of k servers on the cloud. It is common for many cloud-based applications to run on a cluster of servers. Let the cloud request rate seen by the application be λ req/s. If this application were to be deployed at the edge instead of the cloud, we assume that it still uses k servers, but that these servers are distributed across multiple edge sites. In the limit, the application can use k edge sites, each hosting one server for the application. Suppose that the incoming workload of λ req/s is uniformly (equally) distributed across these k edge sites. As shown in Figure 1, each edge site sees a workload λ/k req/s with a service rate of μ. The response time of this queuing system can be proved to be [17]:

$$r_{edge} = \frac{1}{\mu - \lambda/k} \tag{1}$$

In this case, the server latencies at the edge and the cloud can be viewed from the perspective of the well-known bank teller problem that compares a separate queue per teller versus a single queue for all tellers, a problem well studied in the queuing theory literature since the seventies [16, 24, 35].

The main insight of the bank teller problem is that customers will always see lower waiting times when using a single queue for all tellers versus a separate queue per teller. Since the time needed to service each customer varies from customer to customer, in the latter case, some queues see longer waiting times when some customers from those queues make long transactions. A centralized queue avoids such problems since there is a single queue and all queued customers see the same impact. In line with this intuition, queuing theory shows that a centralized queue always yields lower wait time with a few known exceptions (e.g., when jockeying between queues is permitted [24]).

In our case, the centralized queue with k tellers is analogous to the cloud deployment of the application with k servers and a single arrival queue with a total service rate of $k\mu$. A separate queue per teller is analogous to each edge site with a single server that services its own queue. Similar to Equation 1, we can calculate r_{cloud} to be:

$$r_{cloud} = \frac{1}{k\mu - \lambda} = \frac{r_{edge}}{k} \tag{2}$$

This result shows us that at high utilization levels, the queuing delays at the edge can be *factor of k times* higher than the cloud. Thus, even when the workload is equally distributed across edge sites, queuing theory from the bank teller problem tells us that $r_{edge} > r_{cloud}$, since waiting times at individual queues at the edge will be higher than that of the cloud.

Thus the edge has lower network latency than the cloud $n_{edge} < n_{cloud}$, but has higher server latency than the cloud $r_{edge} > r_{cloud}$. If the higher queuing delays at the edge offset the lower network latency, the total end-to-end latency of the edge can actually become higher than that of the cloud. That is $(n_{edge} + r_{edge}) > (n_{cloud} + r_{cloud})$. We refer to this problem as the *performance Inversion problem* of the edge. Such performance inversion only occurs at higher utilization levels, where edge queuing delays are significantly higher than the cloud. Thus, it is important to analyze and quantify the utilization levels above which performance inversion will occur.

This problem can be posed as a variant of the bank teller problem with driving times. A customer can drive to a nearby local bank branch that has a single teller, or they can drive to the main bank branch, which is further away but has k tellers and a single queue. The total time now includes driving time (i.e., network latency) and the time spent at the branch (server latency), which also includes waiting times in the queue. This variant of the bank teller problem with driving times has not been studied previously.

The bank teller problem has been extensively studied in many contexts over the years. Recently, the trade-offs of using centralized versus distributed processing for multi-server applications was studied in [31, 32]. The work showed that the average queue length in steady state scales as a function of the degree of the fraction of centralized servers p and the traffic intensity, λ. However, the potential for edge performance inversion has not been considered by prior research in edge or cloud computing.

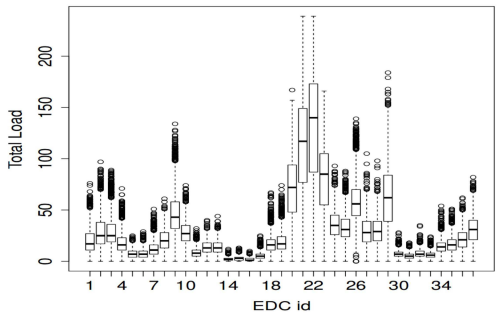


Figure 2: Non-uniform geographic distribution of the total load, measured as the number of vehicles in a cell across time, seen by edge data centers for a vehicular edge application [18].

Impact of workload skews. Our bank teller analogy assumed that the aggregate workload λ is equally partitioned across the k edge sites, yielding a workload of λ/k per edge site. Such balanced workload is the ideal case for the edge. In practice, edge and cloud workloads will exhibit spatial and temporal dynamics that cause load skews. In particular some edge sites may see higher request rates than others due to spatial dynamics, causing the total workload to be split *unequally* across sites. Spatial and temporal dynamics will mean that these imbalances will change over time and also cause the set of edge sites that see higher arrivals to change over time.

These workload skews exacerbate the performance inversion problem, since some edge sites will now see higher waiting times due to higher arrival rates at those sites. Workload skews can result in performance inversion even at lower utilization levels. Note that skews have little impact on the cloud since the cloud still sees the same workload of λ. Workload skews have been observed in empirical studies. For example, Le Tan et al. [18] have shown that load on edge clouds is non-uniformly spatially distributed, using GPS traces of taxis in San-Francisco [23], and assuming each edge cloud serves a hexagonal area of radius 1 km. Figure 2 shows a box-plot of the load seen on each cell based on the data from [23], with some cells having significantly larger loads than others [18]. In other words, while some edge cloud sites will be highly utilized, others will experience low utilization as seen in Figure 2. The load will shift between day and night based on humans' diurnal mobility patterns [7].

The above observations motivates the two principal research questions addressed in this paper

(1) Under what scenarios and utilization levels does the lower network latency of the edge get offset by higher queuing delays, causing a performance inversion where total edge latency becomes worse than the cloud latency?
(2) How do workload skews impact the queuing delays and utilization levels across edge sites and what is its impact on performance inversion at the edge?

These consequences of these questions for application developers and cloud platforms is addressed through a third research question:

(3) How can an application designer reduce the chances of performance inversion for their edge application, and how should a cloud provider ensure its edge cloud customers do not see performance inversion problems?

3 ANALYTIC PERFORMANCE COMPARISON

As discussed above, the edge offers lower network latency than the cloud but can impose higher queuing delays, and thus higher server latency, in some scenarios. In this section, we analytically model the edge and the cloud queuing delays to understand when performance inversion can occur at the edge due to these two opposing factors.

3.1 Queuing Models of Edge and Cloud

Our analytic models of edge and cloud performance are based on queuing theory, a well-known mathematical tool for modeling the waiting time (queuing delays) seen by requests in computing systems. Queuing theory has been extensively used to model cloud performance of web applications [8] in steady-state. and here we use it to compare edge and cloud behavior.

Briefly, queuing theory enables us to model each application as a queuing system. As noted in our bank teller example, we are mainly concerned with multi-server (or clustered) applications that run on a cluster of k servers. In the cloud case, this application is modeled as a queuing system shown in Figure 1b. Incoming requests arrive into a single queue and are scheduled onto one of the k servers based on a certain server scheduling discipline. The model captures an application where requests arrive at a dispatcher node and are scheduled onto one of k cloud servers.

To model the same application when it is deployed at the edge, we assume that the application is distributed across several geographic edge sites and we model each edge site as a separate queuing system as shown in in Figure 1a. The figure shows an example where there are k edge sites, each hosting one server, such that the application still runs on a total of k servers like in the cloud. In this case, the incoming requests get partitioned across edge sites and requests are assumed to be sent to the nearest edge location. Each edge site has a queue for incoming requests and gets scheduled onto the edge servers based on a scheduling discipline. The overall edge performance is then the mean of the performance of the k edge sites. For a fair comparison, we assume that the hardware configuration of the server used in the cloud and the edge is identical.

Formally, application is assumed to run on k cloud servers, $k \geq 1$, and collectively services a mean workload of λ requests/s. When the same application is deployed at the edge, the k servers are distributed across up to k geographic edge site locations and incoming requests are sent to the closest edge site for service. For the ease of analytic modeling, we assume that the application uses k edge locations, each hosting one server. All our results are easily extended to the general case where each site hosts more than one server.

Initially, we assume that the workload is balanced across all edge sites (we relax this assumption and consider the impact of workload skews in §3.3.) Thus, each edge site receives λ/k request/s. Let n_{edge} and n_{cloud} denote the network round trip latency to the edge and the cloud, respectively. Since edge servers are more proximal to end-users, $n_{edge} < n_{cloud}$. The server latency at the edge and the cloud consists of two components: the queuing delay experienced by a request and the service time to execute the request. Let w_{edge} and w_{cloud} denote the queuing delay (i.e., waiting time) incurred by a request upon arrival at the edge and the cloud server, respectively. Also, let s_{edge} and s_{cloud} denote the service time of the request at the edge and cloud, respectively. The end-to-end latency seen by a request is the sum of the network latency, server queuing delay, and the request service time. That is,

$$T_{edge} = n_{edge} + w_{edge} + s_{edge} \tag{3}$$

and

$$T_{cloud} = n_{cloud} + w_{cloud} + s_{cloud} \tag{4}$$

where T_{edge} and T_{cloud} denote the total end-to-end latency of the edge and the cloud, respectively.

In any queuing system, the utilization ρ is given by the ratio of the request arrival rate and the request service rate (i.e., departure rate or system throughput). For example, if a single server has a maximum capacity to service μ requests/s and sees an arrival rate of λ requests/s, then its utilization is given as $\rho = \frac{\lambda}{\mu}$. In our case of the edge, each edge site sees an arrival rate of λ/k. The request execution time is s_{edge}, which means that the server can process $\mu = 1/s_{edge}$ requests/s. Hence $\rho_{edge} = (\lambda/k)/\mu = \lambda/k\mu$. In case of the cloud, the arrival rate is λ. The execution time of a request is same as that of the edge server since the hardware configurations are assumed to be the same. Hence the service rate of a server is μ, same as the edge server. Since there are k cloud servers, the total service rate is $k\mu$. Hence, $\rho_{cloud} = \lambda/k\mu$.

3.2 Modeling Edge and Cloud Latency

Given the above queuing systems, we next analyze the latency seen by the application in the edge and the cloud.

3.2.1 Edge and Cloud Latency Bound. Queuing theory provides well-known closed-form results for waiting times under different workload arrival distributions and different server scheduling disciplines [8]. We can use these well-known results to compare the queuing delays at the edge and the cloud at different utilization levels and derive closed-form equations to determine when higher queuing delays at the edge will cause a performance inversion. Such equations provide easy "rules of thumb" for a system designer or an application developer to analyze whether their edge applications are vulnerable to a performance inversion.

The simplest type of queuing model assumes that the server sees Poisson arrival of requests, and that request service times follow an exponential distribution. In this case, the single server at each edge site is modeled as a $M/M/1$ queuing system, while the k servers at the cloud are collectively modeled as a $M/M/k$ queuing system with FCFS scheduling. $M/M/x$ queuing models have been widely used in computer systems modeling, where the first M denotes that the request arrival rate at the server is assumed to follow a Poisson process, and the second M denotes that the job service times have an exponential distribution. In this formulation, the x is the number of servers in the system. These queuing models yield our first key result.

Lemma 3.1. *The end-to-end latency of the edge is higher than the cloud whenever the network round trip latency difference between the*

edge and the cloud Δn is less than

$$\sqrt{2}\left(\frac{1}{1-\rho_{edge}} - \frac{1}{\sqrt{k}(1-\rho_{cloud})}\right). \quad (5)$$

Proof: The edge will offers worse end to end latency than the cloud when $T_{edge} > T_{cloud}$. That is

$$n_{edge} + w_{edge} + s_{edge} > n_{cloud} + w_{cloud} + s_{cloud} \quad (6)$$

Assuming that both the edge and cloud employ the same hardware configuration for their servers and incoming requests are serviced using the FCFS service discipline, the execution time of a request on an edge server and the cloud server will be identical. That is, $s_{edge} = s_{cloud}$. The above inequality then reduces to

$$n_{edge} + w_{edge} > n_{cloud} + w_{cloud} \quad (7)$$

Let Δn denote the difference in network round trip times between the edge and cloud data centers. That is $\Delta n = n_{cloud} - n_{edge}$. Substituting Δn in the above inequality yields

$$\Delta n < w_{edge} - w_{cloud} \quad (8)$$

This inequality formally states our intuition—that the edge will worse than the cloud when the reduction in network latency at the edge if offset by higher queuing delay at the edge than the cloud.

Our bank teller analogy intuitively tells us that the single queue for k servers at the cloud should yield an overall lower queuing delay at the cloud than that at the edge. We now formally analyze the queuing delays offered by the edge and the cloud. To do so, we use a well-known queuing theory result by Whitt [35], which shows that for the same probability of delay, a system using a multi-server queue requires less resources than one using multiple single server queues. The analysis in [35] also states that the expected waiting time for a request in a system of k servers is

$$E[w|w > 0] = \frac{\sqrt{2}}{(1-\rho)\sqrt{k}}. \quad (9)$$

Applying this result to Equation 8, we have

$$\Delta n < E[w_{edge}|w > 0] - E[w_{cloud}|w > 0] \quad (10)$$

where $E[w_{edge}|w > 0]$ and $E[w_{rem}|w > 0]$ are the expected queuing delays for the edge and the cloud system, given that the queuing delay is non-zero. We note that the accuracy of Whitt's approximation result in Equation 9 increases with higher utilization, since queuing delays are more likely to be non-zero at high utilization levels. Substituting Equation 9, we get

$$\Delta n < \frac{\sqrt{2}}{(1-\rho_{edge})} - \frac{\sqrt{2}}{(1-\rho_{cloud})\sqrt{k}} \quad (11)$$

which completes the proof. □

We now explain the practical implications of the above result by deriving several corollaries and discussing the key takeaways.

Corollary 3.2. *In the case where the application workload is equally balanced across edge sites and the same server configuration is used by the edge and the cloud, the above result reduces to $\Delta n < \frac{\sqrt{2}}{(1-\rho_{edge})}(1 - \frac{1}{\sqrt{k}})$, which yields a bound on the server utilization above which edge performance inversion occurs*

$$\rho_{edge} > 1 - \frac{\sqrt{2}}{\Delta n}(1 - \frac{1}{\sqrt{k}}) \quad (12)$$

Proof: As explained earlier, $\rho_{edge} = (\lambda/k)/\mu = \lambda/k\mu$, and $\rho_{cloud} = \lambda/k\mu$. Since ρ_{edge} is same as ρ_{cloud}, we can substitute ρ_{edge} for ρ_{cloud} in the above lemma, and the result reduces to

$$\Delta n < \frac{\sqrt{2}}{(1-\rho_{edge})}(1 - \frac{1}{\sqrt{k}}) \quad (13)$$

Rearranging ρ_{edge} in this inequality, we get

$$\rho_{edge} > 1 - \frac{\sqrt{2}}{\Delta n}(1 - \frac{1}{\sqrt{k}}) \quad (14)$$

□

Corollary 3.3. *As the number of edge locations k increases, the cutoff edge utilization that yields a performance inversion becomes a function of only Δn.*

Proof: As $k \to \infty$, the term $\frac{1}{\sqrt{k}} \to 0$ in the above inequality, which yields $\rho_{edge} > 1 - \frac{\sqrt{2}}{\Delta n}$ □

Practical takeaways: The above corollaries provide the system designer with a cutoff utilization threshold above which edge performance inversion will occur. Given the choice of an edge deployment that offers network latency n_{edge} or a cloud deployment that offers a network latency n_{cloud}, the above results offers simple rules of thumb for comparing these two deployments—if the expected system utilization will be lower than $1 - \frac{2}{\Delta n}(1 - \frac{1}{\sqrt{k}})$, the edge will indeed provide a lower end-to-end latency. Otherwise the cloud offers a better choice. Further, for a more geographically distributed edge deployment with large k, the utilization needs to be lower than $1 - \frac{2}{\Delta n}$ to avoid a performance inversion.

It is also important to understand the applicability of these results in practice. Our analysis assumed a *multi-server application* that either runs on k servers in the cloud or on a set of geo-distributed servers at k edge sites, where $k > 1$. Consider the special case where $k = 1$, i.e., an application that run at a *single edge site* or one that runs on a single server (which also implies it runs at a single edge site). In either case, edge performance inversion will *never occur*. To see why, when the entire application runs at a single edge site (i.e., is not geo-distributed), the entire cloud workload of the application is also seen by this one edge site. Since the server configuration is identical in both cases, the server latencies at the cloud and the edge are also identical. Hence the edge will always provide better end-to-end response time due to its lower network latency. This can be seen in Corollary 3.2 where substituting $k = 1$, reduces the requirement for performance inversion to $\rho_{edge} > 1$, which can never be true since utilization can not exceed 1. Interestingly, performance inversion can still occur for the case of $k = 1$ if the edge uses a *different server configuration* than the cloud, and specifically if the edge uses more resource-constrained servers (e.g., servers with fewer cores or slower processors). In this case, the edge request execution times are slower than those at the cloud. This results in

higher queuing delays at the edge than the cloud, implying that performance inversion can still occur at the edge for the case of $k = 1$.

The impact of using more resource-constrained servers at the edge also makes performance inversion more likely for multi-server geo-distributed applications (i.e., when $k > 1$). In this case, the execution time of requests are no longer the same at the edge and the cloud, and $s_{edge} > s_{cloud}$. Further performance inversion will occur if $\Delta n < (w_{edge} - w_{cloud}) + (s_{edge} + s_{cloud})$. Since $s_{edge} + s_{cloud} > 0$, the right side of the inequality is larger than our result in Lemma 3.1, implying that performance inversion becomes more likely.

Corollary 3.4. *A hard lower bound on the cloud network latency below which the edge yield worse response time than the cloud is given by*

$$\sqrt{2}\left(\frac{1}{1-\rho_{edge}} - \frac{1}{\sqrt{k}(1-\rho_{cloud})}\right), \tag{15}$$

Proof: The best case for the edge network latency is for a very proximal edge where n_{edge} is nearly equal to zero. Substituting $n_{edge} = 0$ in Δn, the lemma reduces to

$$n_{cloud} < \sqrt{2}\left(\frac{1}{1-\rho_{edge}} - \frac{1}{\sqrt{k}(1-\rho_{cloud})}\right). \tag{16}$$

We can see that if n_{cloud} is lower than the expression on the right, the lemma is always true and the edge will always yield worse end to end latency than the cloud. Hence, the expression is a hard lower bound on the cloud network latency, below which the edge yields worse performance than the cloud. □

Practical takeaways: Since major clouds platforms such as Amazon EC2 and Azure have begun to deploy additional data centers in various geographic regions, the network latency to the cloud is decreasing when applications choose the closest cloud data center. Corollary 3.4 implies that if the cloud network latency drops below a certain threshold through deployment of regional data centers, it has the potential (at certain utilization levels) to offer overall end to end response times that are "good enough" for edge applications and lower than what a smaller edge site can provide. Put another way, with increasing regional cloud data deployments, the benefits of the lower network latency offered even by the best edge deployments will diminish and the cutoff utilization level for performance inversion will also be lower, allowing the cloud to be "good enough" even at lower utilization levels.

3.2.2 Generalized Latency Bounds. Our analysis so far assumed that application request arrivals and service times follow the Poisson and exponential distribution, respectively. This assumption was made for analytical tractability, but real application workloads can have any arbitrary arrival and service time distributions. This variability can be as a result of either the workload dynamics, or the performance variability in cloud systems. We now extend the above analysis to the general case. To do so, we assume each edge site is modeled as a G/G/1 queuing system and the cloud is be modeled as a G/G/k system. In queuing theoretic terminology, G/G/k refers to a queuing system of k servers that service a workload with with a general (i.e., arbitrary) distribution for arrival rates and service times.

Lemma 3.5. *Assuming that the edge and the cloud are modeled as G/G/1 and G/G/k queuing systems, respectively, the edge offers higher end-to-end latency times whenever the network latency difference between the cloud and the edge Δn is less than*

$$\rho_{edge}\frac{1}{\mu_{edge}(1-\rho_{edge})}\frac{c^2_{edge_A} + c^2_{edge_B}}{2} - \frac{\rho^k_{cloud} + \rho_{cloud}}{2}\frac{1}{\mu_{cloud}(1-\rho_{cloud})}\frac{c^2_{cloud_A} + c^2_{edge_B}}{2k}.$$

Proof: Although there are no closed-form equations for waiting times (queuing delays) in a G/G/1 and G/G/k queuing systems, there are several good approximations for waiting times that yield closed-form equations. One such approximation for the expected waiting time is the Allen-Cunneen approximation [3, 12], a widely used queuing theory result that has been shown to be reasonably accurate at higher utilization levels [36], and has found many practical applications [2, 9].

The Allen-Cunneen approximation states that the expected waiting time for a G/G/1 queue is,

$$E(w) \approx \frac{\rho}{\mu(1-\rho)}.\frac{c^2_A + c^2_B}{2}, \tag{17}$$

where, c^2_A and c^2_B are the squared Coefficients of Variation (CoV) of the inter-arrival time and the service times respectively. For G/G/k systems [3, 36], the approximation for the expected waiting time is given as

$$E(w) \approx \frac{P_s}{\mu(1-\rho)}.\frac{c^2_A + c^2_B}{2k}, \tag{18}$$

where P_s is the steady state probability that an arriving request has to wait in the queue for a server to become available. Bolch et al. [3] have shown that P_s can be approximated closely as

$$P_s \approx \begin{cases} \frac{\rho^k + \rho}{2}, & \text{if } \rho > 0.7. \\ \rho^{\frac{k+1}{2}}, & \text{if } \rho < 0.7 \end{cases} \tag{19}$$

Since the Allen-Cunneen approximation is more accurate in higher utilization regimes, and since edge performance inversion is likely at higher utilization levels, we consider the higher utilization case from Equation 19, i.e., $\rho > 0.7$. Since $\Delta n = w_{edge} - w_{cloud}$ as noted in Equation 6, substituting Equation 17, Equation 18 and Equation 19 in Equation 6, we get

$$\Delta n < \rho_{edge}\frac{1}{\mu_{edge}(1-\rho_{edge})}\frac{c^2_{edge_A} + c^2_{edge_B}}{2} - \frac{\rho^k_{cloud} + \rho_{cloud}}{2}\frac{1}{\mu_{cloud}(1-\rho_{cloud})}\frac{c^2_{cloud_a} + c^2_{cloud_B}}{2k} \tag{20}$$

where, $c^2_{cloud_A}$, $c^2_{cloud_B}$, $c^2_{edge_A}$ and $c^2_{edge_B}$ are the squared CoVs of the inter-arrival times and the service times of requests for the cloud and the edge , respectively. Like before, we assume that the edge and the cloud use the same hardware configuration for servers and FCFS service disciplines. Hence the service times of requests on the edge and cloud are the same. Since service times are inverse of the service rate, we have $s_{cloud} = s_{edge} = 1/\mu_{cloud} = 1/\mu_{edge}$. For the same reason, the CoV of service times on the edge and cloud

servers are the same: $c^2_{cloud_B} = c^2_{edge_B}$. Equation 20 then simplifies to

$$\Delta n < \rho_{edge}\frac{1}{\mu_{edge}(1-\rho_{edge})}\frac{c^2_{edge_A}+c^2_{edge_B}}{2} - \frac{\rho^k_{cloud}+\rho_{cloud}}{2}\frac{1}{\mu_{edge}(1-\rho_{cloud})}\frac{c^2_{cloud_A}+c^2_{edge_B}}{2k}, \quad (21)$$

□

Corollary 3.6. *As the number of edge locations k increases Δn becomes solely a function of the edge workload parameters*

$$\Delta n < \rho_{edge}\frac{C_{edge}}{\mu_{edge}(1-\rho_{edge})}, \quad (22)$$

where C_{edge}, is a variability measure and is equal to $\frac{c^2_{edge_A}+c^2_{edge_B}}{2}$

Proof: As $k \to \infty$, the term $\frac{c^2_{cloud_A}+c^2_{edge_B}}{2k} \to 0$, and so does the second term of the inequality in Equation 21, yielding the above inequality. □

Practical takeaways Unlike our M/M/k results which depended solely on the server utilization levels, our general results indicates that edge performance inversion depends both on the utilization levels as well as the burstiness of the workload , which is captured by the coefficient of variation (CoV) of arrival rates and service times.

Corollary 3.6 implies that spikes, flash crowds, and high variability in the edge workload inter-arrival (and service) times can have a significant impact on whether an edge will see a performance inversion. When the workload is bursty in nature, the CoV of the inter-arrival times will be highly variable. This makes performance inversion more likely and implies that edge is less suitable for applications where the request rates (and hence the inter-arrival times) have higher burstiness.

3.3 Impact of Workload Skews

Our analysis thus far assumed equally partitioned edge workload across edge sites. However, application workloads are unlikely to be geographically balanced and exhibit spatial skews (as shown in Figure 2). For example, an edge application such as an online game may be more popular in certain parts of the world than others causing the aggregate workload to be split unequally across geographic sites. We extended our analysis to handle such spatial imbalances in the workload. Let the total workload of λ req/s be arbitrarily partitioned across the k edge sites with edge site i receiving λ_i req/s. Across all edge sites $\sum_i \lambda_i = \lambda$. The fraction of the total workload seen by edge site i is given by $w_i = \lambda_i / \sum_j \lambda_j$. Since each edge site sees a different fraction w_i of the requests, the utilization and queuing delays of each edge site will be different. Specifically in Lemma 3.1, the waiting time seen by requests at site i will be

$$E[w_i l w_i > 0] = \frac{\sqrt{2}}{1-\rho_{edge_i}} \quad (23)$$

where $\rho_{edge_i} = \lambda_i/\mu$.

The average queuing delays seen across the edge is the weighted average of the queuing delays seen by each site. That is, $\sum w_i \sqrt{2}/(1-\rho_{edge_i})$ Hence, Lemma 3.1 can be restated as follows.

Lemma 3.7. *In the scenario where the workload sees spatial skews and is partitioned unequally across edge site, the edge will offer worse end-to-end latency when*

$$\Delta n < \sqrt{2}\left(\sum_i \frac{W_i}{1-\rho_{edge_i}} - \frac{1}{\sqrt{k}(1-\rho_{cloud})}\right) \quad (24)$$

Practical takeaways: When the workload exhibits spatial or geographic skews, Equation 23 in the above lemma indicates that sites with higher workloads will experience higher queuing delays. In such cases, the application should no longer deploy an equal number of servers at each edge site—edge locations that see higher workloads should be provisioned with higher processing capacity (in terms of bigger servers or more servers). Specifically, each each edge site should be allocated server capacity in proportion to the workload seen by that site. Further, if the spatial distribution of the workload changes over time, the allocated processing capacity at each site should also be adjusted dynamically to match these workload changes. So long as the processing capacity at each edge site matches the incoming workload, the edge sites will be balanced in their utilization levels, and condition in Lemma 3.7 reduces to that in Lemma 3.1. It is important to note that even with the processing capacity of an edge site is matched to the spatial skews in the workload, the performance inversion problem does not go away. The edge can still yield worse end-to-end latency it the condition of Lemma 3.1 is satisfied

4 EXPERIMENTAL COMPARISON

Having analytically compared edge and cloud latencies from the perspective of performance inversion, we now experimentally compare the two using a realistic application and real cloud workloads.

4.1 Experimental Setup

Edge/Cloud deployments. To demonstrate that edge performance inversion can occur in real world settings, we conduct our experiments on Amazon's EC2 cloud. The EC2 cloud platforms offers numerous choices of cloud data center locations which allows us to conduct experiments on cloud servers with different network latencies. We utilize these choice of data center locations to experiment with different edge and cloud application scenarios with different network latencies. All our experiments assume an edge round trip latency of 1ms, which represents the best case scenario for edge deployments. We do so by placing the end client (emulated using a workload generator) and the edge servers in two different availability zones (data centers) within the same EC2 availability region, yielding a very low user-to-edge network latency. We use four different cloud deployments to complement our 1ms edge deployment.

Our first scenario assumes a *nearby* cloud that is around 15ms away, yielding a small Δn ($\Delta n < 20$). As the number of cloud data centers has grown, it is now feasible for a cloud server to have less than 20ms round trip time (RTT) latency from the user. We place the end-client and the edge servers in the us-east-2 EC2 region (in Ohio), and the cloud servers in the us-east-1 in Virginia, with an average Round-Trip-Time (RTT) of around 15 ms. [1]

[1]We also ran similar experiments with the edge in Ireland and the Cloud in London, but omit these results since they are similar.

Our second scenario assumes a cloud that is between 20 to 30 ms away, yielding a medium Δn ($\Delta n \sim 20$). This is an increasingly common latency seen by end-users in populated urban centers. Our experiments use two configurations. We use an edge deployment in Ireland, and a cloud deployment in Frankfurt, yielding a network RTT between 20 to 24 ms. We also use an edge deployment in us-east-2 (Ohio),and a cloud deployment in ca-central-1 (Montreal) with an RTT time between 25 to 28 ms.

Our third scenario assumes a cloud that is located over 50ms away, yielding a larger Δn ($\Delta n > 50$). Users in many parts of the world may still experience cloud RTTs that exceed 50ms. It is also common to experience higher RTT latencies when using a cellular connection. In this case, we deploy the edge and end-clients in us-east-2 (Ohio) and the cloud in us-west-1 (North California), with an average RTT of around 50 to 60 ms.

Finally, we also experiment with a distant cloud application by deploying the edge in the US (us-east-1), and the cloud in Europe (Ireland), with latencies exceeding 80ms. Since this involves crossing trans-continental ocean links, well-designed geo-distributed applications will typically avoid such latencies but we include this scenario for completeness.

All experiments assume the same configuration for the cloud and edge servers; we use the c5a.xlarge instance type, with 4 vCPUs, 8 GB of memory, and 10GB/s network inferences. We assume that the cloud deployment of the applications runs on 5 or 10 servers in the cloud, while corresponding edge deployments use one or two servers at each site. In the cloud case, we use HAProxy, a popular load-balancer to balance the load across the different cloud servers.

Application Workload Since many types of edge applications such as mobile AR, visual analytics and IoT processing involve some form or machine learning, we choose deep neural network (DNN) inference workload for our experiments. The application is a web-based DNN image classification service built using Keras, TensorFlow, and Flask that is designed to receive image requests from an end-client and responds with the class of the image. Note that this is more compute-intensive workload than simple web applications.

We built a workload generator using Gatling, an open-source load- and performance-testing framework based on Scala. We choose Gatling for its scalability. We use a large images dataset from Kaggle. Each second, the workload generator randomly selects a set of images, based on the number of requests configuration for the experiment, and sends them to the edge application for classification. When the response is received, the workload generator logs the end-to-end response time. The workload generator is capable of generating requests that follow a specified distribution or it can replay a request trace.

Azure Trace Workload In addition to synthetic workload generation, we use traces from the Azure Public Dataset [28] for our evaluation. This dataset provides request traces seen by serverless functions that run on the Azure cloud. To construct the edge site specific workload from these traces, we first choose a set of functions that belong to the same applications and group them into k mutually exclusive sets. The request traces for each grouping of functions is then mapped onto one edge site. Figure 8 depicts the workload seen by five edge sites that we construct from the aggregate trace and shows spatial and temporal variations that are present in the trace. The cloud trace is then the aggregated request trace from all edge sites. In addition, the Azure dataset provides execution times of serverless functions as coarse-grain distributions; we sample these distributions to generate an execution time and append it to each request in the trace. When replaying the trace, an image of an appropriate size is chosen to generate a request with the appropriate service time.

Research questions Our experimental comparison is designed to answer several research questions: (1) How likely is the edge performance inversion problem in real-world cloud and edge setting under realistic workloads? (2) What types of edge utilization levels result in a performance inversion? (3) How well do our experimental results validate the predictions of our analytic models? (4) How do tail response times of the edge compare with the cloud? (5) How do cloud locations with different network latencies impact our results?

4.2 Mean Latency Comparison and Validation

First we benchmark our application while running on the c5a.xlarge EC2 instance type. Since DNN inference requests are compute intensive, we find that the system reaches 100% utilization at 13 req/s where it starts dropping requests or thrashing. Hence our experiments assume 12 req/s to be the maximum practical sustainable workload on a c5a.xlarge instance, which yield utilization levels of around 90%.

Our first experiment uses our typical cloud setup with the edge in Ireland and the cloud in the Frankfurt EC2 region. We vary the request rate at each edge server from 6 to 12 req/s and measure the mean end-to-end request latency. We compare this edge setup to a cloud setup with 5 servers that see the cumulative request rate of 5 edge sites (i.e., when each edge server sees n reqs/s, the cloud servers see $5n$ req/s). We also experiment with an edge deployment with 2 servers per edge site and compare it to a cloud deployment with 10 servers. Figure 3 plots the mean end-to-end latency seen for the edge and the cloud setups. As can be seen, the application provides lower response times from the edge at lower request rates. As the request rate is increased, the edge latency increases faster than the cloud latency and there is crossover point at 8 req/s where the edge latency becomes higher than the cloud latency, causing a performance inversion beyond this workload level. A similar behavior is seen for the case where the edge has two servers per site and the cloud has 10 servers, but the cross over occurs at a higher workload of around 11 req/s.

These results show that the performance inversion can occur even in very low latency edge environments (e.g., 1ms network round trip latency) and at moderate utilization levels of $\rho = 8/13 = 0.61$. Our corollary 3.2 predicts a cutoff utilization of $\rho_{edge} = 0.64$ for $\Delta n = 30$ and $k = 5$, which is within 4.5% of the experimentally observed value. For the two server edge case and $k = 10$, our analytical result predict a cutoff utilization of $\rho_{edge} = 0.75$, which is within 6% of the measured value.[2] Thus, our analytic models can also predict the performance inversion utilization threshold with good accuracy.

[2]Some of this model error is a result of the discrete changes in ρ caused by discrete changes in the request rate as it is varied from 1 to 12 req/s.

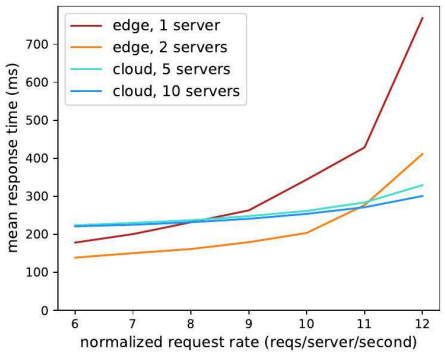


Figure 3: Mean latency of edge (Ireland) and a typical cloud (Frankfurt, ~25ms)

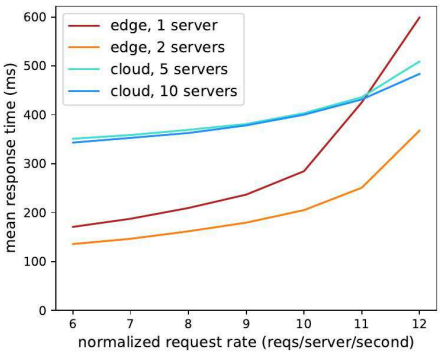


Figure 4: Mean latency of edge (Ohio) and distant cloud (N California, ~54ms)

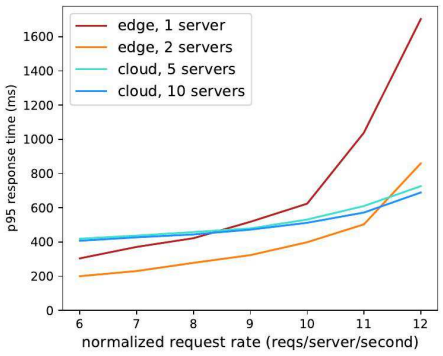


Figure 5: Tail latency of edge (Ohio) and distant cloud (N. California, ~54ms)

4.3 Tail Latency Comparison

There has been considerable research on reducing tail latencies of cloud applications to improve overall end-user experience [19, 29]. While our analytical results only permit a comparison of mean latencies of the edge and the cloud, we can use experimentation to compare tail latencies as well as the distributions of the edge and cloud latencies. To do so, we repeat the previous experiment with an edge deployment in Ohio and the cloud deployment in N. California. Note that unlike the previous experiment where the cloud was 20-30ms away, the cloud deployment here is 50-60ms away making a 1ms edge even more beneficial from a network latency perspective. Figure 4 plots the mean latencies of the edge and the cloud deployments. Since the cloud is more distant, the figure shows that edge latencies are lower than the cloud for a wider range of utilization values. For the 5 cloud server case, a performance inversion is seen at 11 req/s (i.e., 84.6% utilization). For the 10 server cloud deployment, we do not see an inversion even at 12 req/s, indicting the inversion occurs close to saturation rate of 13 req/s. Not surprisingly, when the cloud is more distant, edge performance inversion is less likely and will occur at higher utilization level.

Figure 5 plots the tail latency of the edge the cloud, where tail latency is defined to be the 95-th percentile of the end-to-end latency distribution. The figure reveals an interesting insight that goes beyond our analytical results. It shows that when tail latencies are concerned, performance inversion occurs a *much lower utilization* level than that for the mean latency. Further, even when the edge is well-behaved (i.e., offers lower mean latency than the cloud), it can still see a performance inversion with respect to the tail latency. The figure shows that performance inversion occurs at 8 req/s (61% utilization) for the 5 cloud server case and for the case of 10 cloud servers with 2 servers per edge site, it occurs at 11 req/s (84% utilization). Note that at these level, the mean edge latency is lower than the cloud latency and there is no inversion with respect to the mean. Figure 6 shows violin plots of distributions of the end-to-end latency seen at the edge and the cloud for 10 req/s. As can be seen, edge requests see a higher variability in the end-to-end latency than cloud requests, and the latency distribution at the edge has a longer tail than the cloud.

4.4 Impact of Cloud Locations

We next study the impact of varying the cloud latency of the application by deploying the application at various cloud locations and measure the cutoff utilization where the edge latency becomes worse than the cloud. Figure 7 plots cutoff utilizations for the mean and tail latency for various cloud deployments (the edge is 1ms away in all cases). The figure shows that as the cloud gets closer to the user, the cutoff utilization for a performance inversion keeps decreasing. For a 15ms cloud (US-east-1), the cutoff utilization for the mean is only 40% and that for the tail latency is even lower at 25%. For a 25-30ms cloud, the cutoffs are 60% and 40% respectively. For a transcontinental cloud (80ms RTT), the cutoff for the mean is close to saturation but that for the tail is 75%. The figure shows that as cloud deployments increase, edge performance inversion can occur at progressively lower utilization levels, making it more likely in real-world deployments.

4.5 Comparison using Azure Workloads

Our final experiment involves replaying trace workloads seen at real cloud, i.e., using Azure public cloud traces. The experiment emulates generalized distributions for arrival rates and service times and also includes dynamic spatial and temporal workload skews, as shown in Figure 8. We replay the trace workload shown in Figure 8 at each of the five edge sites and the cumulative workload to the cloud site with 5 servers. We use the typical case cloud scenario with the edge at US-east-2 (Ohio) and the cloud in Montreal (ca-central-1), with RTT of 25 to 28ms. Figure 9 compares the mean latency seen by requests across all edge sites to the mean cloud latency. The temporal fluctuations across sites causes the utilization of each edge site to vary over time and edge sites frequently see a performance inversion, causing the mean edge latency to rise above the cloud latency. The cumulative workload at the cloud sees a smoothing benefits and there is less variation in the cloud latency as a result. Figure 10 shows a box plot of the latency seen by each of the five edge sites and the cloud. The figure show the unequal partitioning of the workload across edge sites causes different edge sites to exhibit different latencies; the more bursty the workload, the more variable the latency distribution. Edge site 4 see a lower workload than the rest and also offers the lowest latencies as a result.

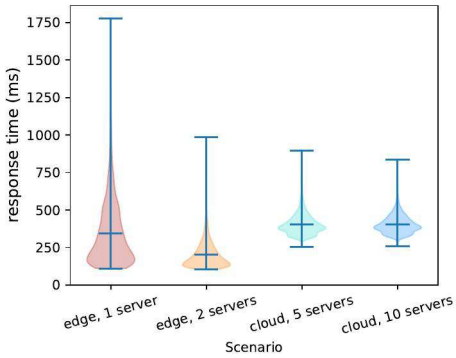


Figure 6: Response distribution of edge (Ohio) and distant cloud (N. California) for 10 req/server/s.

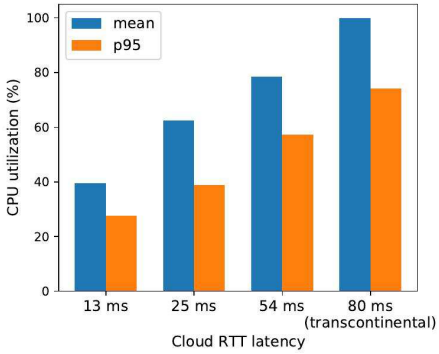


Figure 7: Utilization levels above which the edge provides worse mean and tail latencies for various cloud settings.

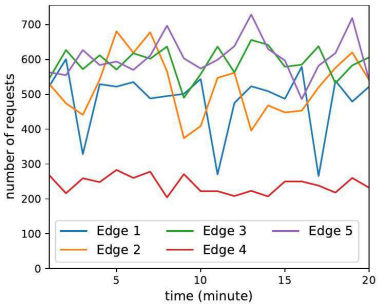


Figure 8: Edge workload of five edge sites based on the Azure serverless traces.

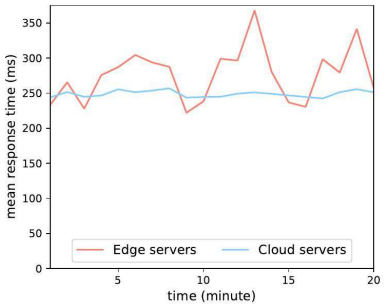


Figure 9: Mean edge and cloud latencies for Azure trace workload.

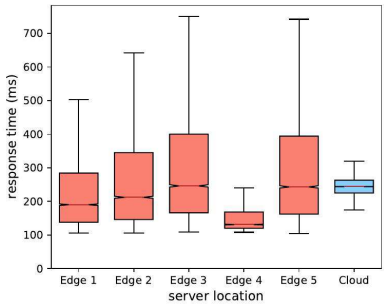


Figure 10: Higher tail latency of the edge under the Azure workload.

4.6 Impact of Server Heterogeneity

Our experiments thus far used a single server type, namely c5.xlarge servers. To demonstrate that edge performance inversion can occur under a range of server types, we repeat our experiments with 2-core c5.large, 4-core c5.xlarge, 8-core c5.2xlarge servers at both the cloud and the edge. We also conduct experiments with the g4dn.xlarge GPU-enabled servers where both the CPU and the NVIDIA T4 GPU are used for processing each request. Finally, we use a heterogeneous deployment with GPU and CPU server types, where half the cluster servers are g4dn.xlarge GPU servers and the other half are c5a.xlarge servers in both the edge and cloud. In all cases, the total number of servers k in the edge and the cloud is kept the same. Figure 11 shows the cutoff per-server request rate at which edge performance inversion is seen for each configuration. The figure shows that regardless of the server configuration, the edge always sees a performance inversion after a cutoff request rate. Not surprisingly, as the servers become more capable (e.g., have more cores or have a GPU), the per-server request rate at which inversion occurs also rises. Although we omit cutoff utilizations levels due to space constraints, the shown cutoff request rates correspond to utilizations levels ranging from 59% to 65.8% across various configurations. Interestingly, even hybrid deployments with a mix of CPU and GPU servers types at each cloud or edge sites see a similar inversion behavior.

4.7 Performance of Single Server Applications

Our analytic models assume a multi-server clustered application with k servers and also assume identical configuration for cloud and edge servers. Neither assumption may hold in some scenarios since (i) the application may be a "small" single server application, and (ii) the edge may be more constrained than the cloud (an architecture argued by many edge researchers). We conduct an experiment using a single server application ($k = 1$) that runs in the cloud and then at a single edge site. We first use the same c5.2xlarge server configuration in the cloud and the edge, and then make the edge progressively more constrained by using c5.xlarge and c5.large servers at the edge. Figure 12 shows the edge and cloud response times under varying request rates. First, under identical server configurations, a single server application ($k = 1$) does not experience any performance inversion—consistent with Corollary 3.2 which indicates utilization ρ_{edge} will need to exceed 1 for $k = 1$ (implying inversion is impossible). However, as the edge servers gets more constrained, performance inversion is seen at utilizations as low as 30%. This shows that even single server applications see the same performance inversion problem as clustered applications, but only in constrained edge settings.

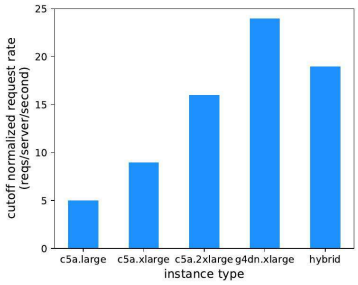


Figure 11: Edge performance inversion occurs for a range of server configurations.

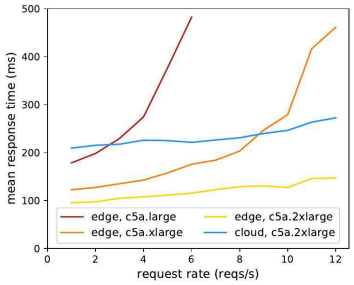


Figure 12: Single server applications see inversion in constrained edge settings.

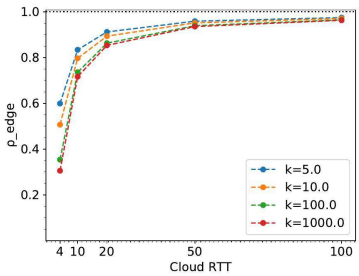


Figure 13: Pareto frontier for Equation 14 with a large cloud

5 DESIGN IMPLICATIONS

We now discuss how our results can be used by application developers and cloud platforms to avoid edge performance inversion.

5.1 Summary of our results.

We summarize our results in the context of the three research questions posed in §2.2. We first visually summarize our results using Pareto curves in Figure 13. The figure shows the cutoff utilization level for various cloud latencies and various application cluster sizes (k) above which an edge deployment will see performance inversion and show these cutoffs for a range of realistic utilization levels and cloud latencies. We note that this Pareto front does not take into account any workload or service time variability, so it shows more of a best-case scenario for edge clouds.

Overall, our analytic results in Lemmas 3.1 to 3.5 show that performance inversion is possible under any arbitrary workload and service time distribution. These lemmas also provide closed-form equations for cutoff utilizations for edge performance inversion for any given deployment. Our experimental results confirm our analytical results and show that edge performance inversion does occur in real-world edge and cloud deployments. Specifically, even a low latency 1ms edge can experience a performance inversion at moderate utilization levels of 40 to 60%. Our experimental results go beyond our analytical equations, and show that even when the edge is well-behaved with respect to mean latency, it can still experience a performance inversion for *tail* latencies. Further, our results hold under a variety of server configurations, including GPU and mixed configurations, and under real-world public cloud workloads. With increasing workload skew or cloud penetration, edge performance inversion can occur at progressively lower utilizations, making it more likely in real-world edge deployments. Similarly, a resource-constrained edge is more likely to experience performance inversion problems. Finally, our results focus on multi-server (clustered) applications, but we also show inversion can occur for single server application in constrained edge settings.

5.2 Application Considerations

Application developers will need to develop new edge resource management techniques, such as new autoscalers or latency control methods, to avoid edge performance inversion issues, When doing so, an application developer can use our results in three different ways: (i) estimate the chances of a performance inversion for a particular edge deployment, (ii) provision additional capacity at the edge site to eliminate or reduce the chances of a performance inversion, and (iii) employ run-time techniques such as geographic load balancing.

Estimate Chance of a Performance Inversion An application designer can use the edge versus cloud latency of a particular setup to determine the cutoff utilization for performance inversion to occur (e.g., using Corollaries 3.2 and 3.3). Typical cloud application are provisioned for the peak workload, and it is well known that they see low average utilization. Assuming that edge applications are also provisioned similarly, we should expect edge sites to also have low utilization levels on average. If the estimated edge utilization is lower than the cutoff utilization, then the chance of edge performance inversion is likely to be small.

Provision extra capacity: One approach for avoiding edge performance inversion is to provision extra server capacity at each site. In this case, the aggregate server capacity at all edge sites is greater than the k servers of the cloud deployment. One simple rule of thumb to guide the amount of overprovisioning at the edge is to use Lemma 3.1. Suppose edge site i has K_i servers and receives λ_i req/s. Then for each site i, Lemma 3.1 yields

$$\Delta n < \sqrt{2}\left(\frac{1}{\sqrt{k_i}(1-\frac{\lambda_i}{\mu k_i})} - \frac{1}{\sqrt{k}(1-\frac{\lambda}{\mu k})}\right) \tag{25}$$

Since Δn, λ_i, and μ are all measurable or can be estimated, we have a numerical lower bound on k_i at each site to avoid performance inversion. An overprovisioning factor can be applied to k_i to allow sufficient headroom capacity to minimize the probability of performance inversion.

Geographic Load Balancing. The bank teller analogy of Section 2.2, which is the basis for edge performance inversion, does not hold if "queue jockeying" [24] is allowed where a customer is allowed to switch queues. In the edge case, this amounts to request redirection to a different edge site if the local edge site is experiencing high waiting times or high utilization. Content delivery networks have employed such geographic load balancing methods to avoid overloading a local edge site [22]. Edge performance inversion can be avoided by employing similar geographic load balancing methods, where requested to an overloaded edge sites are redirected to nearby edge sites with space capacity.

5.3 Edge Cloud Platform Considerations

The edge performance inversion problem has also important implications for edge cloud providers. Each edge site will host applications belonging to multiple customers, each of which see a time varying workload. Like any cloud, an edge cloud has to be provisioned with enough server capacity to serve peak demand across customers. This is done by deploying enough servers to serve a high percentile of the workload across all customers. If the workload arrivals are Poisson, the well known two sigma rule can be used to approximate the 95th percentile of the workload. The two sigma rule states that the 95th percentile of the workload is $\lambda + 2\sigma$, where σ is the standard deviation of the workload. For Poisson arrivals, the standard deviation is the square root of the mean, i.e., $\sigma = \sqrt{\lambda}$. Hence, the cloud requires an aggregate server capacity $C_{cloud} = \lambda + 2\sqrt{\lambda}$ to serve the peak workload. However, for edge sites, and in case of a spatially perfectly balanced workload, each edge site sees λ/k req/s. Hence, each edge site requires enough capacity to serve the peak workload of $\lambda/k + 2\sqrt{\lambda/k}$. Since there are k edge sites, the total edge capacity is $C_{edge} = k\left(\frac{\lambda}{k} + 2\sqrt{\frac{\lambda}{k}}\right) = \lambda + 2\sqrt{k\lambda}$. It follows that $C_{edge} > C_{Cloud}$ since $\sqrt{\lambda} < \sqrt{k\lambda}$.

This means That even for the simplest type of workload (Poisson arrival, balanced across sites), the peak capacity at the edge is higher than that of the cloud. Intuitively, this is due to the statistical smoothing benefits at the cloud that accrue when multiple edge site workload is aggregated at the cloud (which sees a lower peak than the sum of the edge peaks) To avoid its customers from seeing performance inversion, the degree of overprovisioning at the edge has to be even higher than the above analysis (which did not consider the impact of query delays during peak utilization). Our results in Section 3 only considered the average workload λ, but not the impact of peak workload $\lambda + 2\sqrt{\lambda}$ where queuing delays will be even higher. This implies that cloud providers will incur a higher cost to serve N customers at the edge than the cloud and will need to overprovision their edge deployments even more than their cloud deployments.

6 CONCLUSIONS

In this paper we presented and studied the edge performance inversion problem where higher edge queuing delays due to resource constraints or workload skews offset the lower network latency of the edge when compared to the cloud. We analytically compared edge and cloud latencies using queuing models and analyzed conditions under which the edge can yield worse performance than the cloud. We conducted a detailed experimental performance comparison of the edge and cloud performance using realistic applications and Azure trace workloads on real cloud and edge platforms. Our experiments showed that the mean and tail latencies seen by edge applications can indeed be worse than the cloud even at moderate utilization levels of 40 to 60%. We discussed implications of our results for application designers and cloud service providers and provided insights into how such performance inversion problems can be avoided or mitigated. These insights also point to several directions for future work. We plan to design dynamic edge resource allocation techniques that are robust to performance inversion and can optimize edge tail latencies. We also plan to study the economic costs of edge deployments resulting from the need to deploy extra capacity to prevent performance inversion.

Acknowledgements: We thank the anonymous reviewers for their helpful comments. This research was funded in part by NSF grants 2105494, 1908536, 1836752, 1763834, US Army contract W911NF-17-2-0196, Chalmers ICT-AoA, and Amazon AWS cloud credits.

REFERENCES

[1] Azure IoT edge. https://azure.microsoft.com/en-us/services/iot-edge/.
[2] F. Ahmad and T. Vijaykumar. Joint optimization of idle and cooling power in data centers while maintaining response time. In *ACM Sigplan Notices*, volume 45, pages 243–256. ACM, 2010.
[3] G. Bolch, S. Greiner, H. De Meer, and K. S. Trivedi. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.
[4] D. Bruneo. A stochastic model to investigate data center performance and qos in iaas cloud computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):560–569, 2014.
[5] A. Gandhi, S. Doroudi, M. Harchol-Balter, and A. Scheller-Wolf. Exact analysis of the m/m/k/setup class of markov chains via recursive renewal reward. In *ACM SIGMETRICS Performance Evaluation Review*, volume 41, pages 153–166. ACM, 2013.
[6] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. A. Kozuch. Optimality analysis of energy-performance trade-off for server farm management. *Performance Evaluation*, 67(11):1155–1171, 2010.
[7] M. C. Gonzalez, C. A. Hidalgo, and A.-L. Barabasi. Understanding individual human mobility patterns. *nature*, 453(7196):779, 2008.
[8] M. Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*, chapter 14. Cambridge University Press, 2013.
[9] Y.-J. Hong, J. Xue, and M. Thottethodi. Dynamic server provisioning to minimize cost in an iaas cloud. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 147–148. ACM, 2011.
[10] J. Hsu. How youtube led to google's cloud-gaming service: The tech that made youtube work everywhere promises to do the same for games-[news]. *IEEE Spectrum*, 56(09):9–10, 2019.
[11] W. Hu, Y. Gao, K. Ha, J. Wang, B. Amos, Z. Chen, P. Pillai, and M. Satyanarayanan. Quantifying the impact of edge computing on mobile applications. In *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems*, APSys '16, pages 5:1–5:8, New York, NY, USA, 2016. ACM.
[12] S. Kelly-Bootle and B. W. Lutek. Chapter 5 - queueing theory. In A. O. Allen, editor, *Probability, Statistics, and Queuing Theory with Computer Science Applications (Second Edition)*, Computer Science and Scientific Computing, pages 247 – 375. Academic Press, San Diego, second edition edition, 1990.
[13] A. Khan, X. Yan, S. Tao, and N. Anerousis. Workload characterization and prediction in the cloud: A multiple time series approach. In *2012 IEEE Network Operations and Management Symposium*, pages 1287–1294. IEEE, 2012.
[14] H. Khazaei, J. Misic, and V. B. Misic. Performance analysis of cloud computing centers using m/g/m/m+ r queuing systems. *IEEE Transactions on parallel and distributed systems*, 23(5):936–943, 2012.
[15] H. Khazaei, J. Mišić, V. B. Mišić, and S. Rashwand. Analysis of a pool management scheme for cloud computing centers. *IEEE Transactions on parallel and distributed systems*, 24(5):849–861, 2013.
[16] J. Kingman. Inequalities in the theory of queues. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 102–110, 1970.
[17] L. Kleinrock. Theory, volume 1, queueing systems, 1975.
[18] C. N. Le Tan, C. Klein, and E. Elmroth. Location-aware load prediction in edge data centers. In *Fog and Mobile Edge Computing (FMEC), 2017 Second International Conference on*, pages 25–31. IEEE, 2017.
[19] J. Li, N. K. Sharma, D. R. Ports, and S. D. Gribble. Tales of the tail: Hardware, os, and application-level sources of tail latency. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–14. ACM, 2014.
[20] C. Lu, K. Ye, G. Xu, C.-Z. Xu, and T. Bai. Imbalance in the cloud: An analysis on alibaba cluster trace. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2884–2892. IEEE, 2017.
[21] Y. Mao, J. Zhang, S. Song, and K. B. Letaief. Power-delay tradeoff in multi-user mobile-edge computing systems. In *IEEE Global Communications Conference (GLOBECOM),*, pages 1–6. IEEE, 2016.
[22] E. Nygren, R. K. Sitaraman, and J. Sun. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*, 44(3):2–19, 2010.
[23] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser. CRAWDAD data set EPFL/mobility. https://crawdad.org/epfl/mobility/20090224/.

[24] M. H. Rothkopf and P. Rech. Perspectives on queues: Combining queues is not always beneficial. *Operations Research*, 35(6):906–909, 1987.
[25] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal communications*, 8(4):10–17, 2001.
[26] M. Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.
[27] M. Satyanarayanan, V. Bahl, R. Caceres, and N. Davies. The case for VM-based cloudlets in mobile computing. *IEEE pervasive Computing*, 2009.
[28] M. Shahrad, R. Fonseca, Í. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, pages 205–218, 2020.
[29] L. Suresh, M. Canini, S. Schmid, and A. Feldmann. C3: Cutting tail latency in cloud data stores via adaptive replica selection. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pages 513–527, 2015.
[30] L. Tong, Y. Li, and W. Gao. A hierarchical edge cloud architecture for mobile computing. In *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*, pages 1–9. IEEE, 2016.
[31] J. N. Tsitsiklis and K. Xu. On the power of (even a little) centralization in distributed processing. *ACM SIGMETRICS Performance Evaluation Review*, 39(1):121–132, 2011.
[32] J. N. Tsitsiklis and K. Xu. On the power of (even a little) resource pooling. *Stochastic Systems*, 2(1):1–66, 2012.
[33] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung. Dynamic service migration and workload scheduling in edge-clouds. *Performance Evaluation*, 91:205–228, 2015.
[34] M. Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.
[35] W. Whitt. Understanding the efficiency of multi-server service systems. *Management Science*, 38(5):708–723, 1992.
[36] W. Whitt. "approximations for the gi/g/m queue". *Production and Operations Management*, 2(2):114–161, 1993.
[37] J. Xue, R. Birke, L. Y. Chen, and E. Smirni. Spatial–temporal prediction models for active ticket managing in data centers. *IEEE Transactions on Network and Service Management*, 15(1):39–52, 2018.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

We present the toolset used in this research as a general and extensible performance benchmarking framework for real-world edge/cloud environments. Our framework, which is built based on a variety of mature open-source technologies, enables automatic infrastructure provisioning, workload deployment, benchmark execution, and metrics collection.

Our performance benchmark uses a client/server architecture: the client side is one node running the load generator, while the server side is a cluster which consists of one load-balancer node and multiple worker nodes. We use Gatling (https://gatling.io/), an open-source load testing tool, as the load generator. We implemented two load patterns using Gatling for the evaluation in this paper: constant rate Poisson processes, and emulations using Azure Functions Traces (available at https://github.com/Azure/AzurePublicDataset). Gatling also support other distributions such as evenly distributed requests or requests with ramp-up arrival rate. More complex patterns can also be implemented by either 1) describing the simulation procedures using Scala scripts, or 2) recording user behaviors using a bundled recorder and replaying the recorded scenarios during benchmark experiments.

On the server cluster side, each worker node runs an instance of the deep learning REST API server based on Keras and TensorFlow. We use docker as the server runtime to make it easier to change the server application to any other application. For the the load-balancer we use HAProxy running in multi-thread mode to handle high throughput. After receiving requests from the workload generator, HAProxy will distribute the requests among available worker nodes using the lead connection load balancing algorithm in order to maximize resource multiplexing.

For users who want to run the benchmark in a public cloud, we also provide automation scripts based on Terraform for describing and provisioning the desired infrastructure. Although we currently only implement the scripts for AWS EC2, Terraform has support for most major cloud providers, such as Microsoft Azure and Google Cloud, so the scripts can be easily modified to work with other providers. The configurations of all the nodes used in the experiment, including regions/availability zones, instance types, SSH key pair, and the number of worker nodes, can be specified in the scripts. Once the scripts have been properly configured, a user can just use one single command to set up or tear down the testing infrastructure, as demonstrated in the sample workflow below.

Once the testing infrastructure is set up, our framework provide a set of Ansible playbooks for automating the deployment of the performance benchmark. All the aforementioned software components used for the benchmarking experiments will be installed and configured using Ansible. Users can change these playbooks to modify the software environment to their needs. For example, to change the server application running on the worker nodes, a user only need to change a few parameters in the docker_image and docker_container modules in the worker setup playbook. In addition, the experiment execution and data collection are also implemented as playbooks to improve efficiency. For instance, users can run multiple benchmarking scenarios at once to fully explore the performance characteristics, or collect CPU utilization data from all the worker nodes in parallel.

The workflow of a typical run of the benchmarking framework looks like follows:

terraform apply -var load-generator-region=us-east-2 -var cluster-region=ca-central-1 -var worker-count=10 -auto-approve

(cd ansible && ansible-playbook ping.yml && ansible-playbook setup.yml && ansible-playbook experiment.yml)

terraform destroy -var load-generator-region=us-east-2 -var cluster-region=ca-central-1 -var worker-count=10 -auto-approve

Thanks to Gatling, after the benchmark experiment is finished an HTML report will be generated for each simulation with many statistics and beautiful visualizations, such as the response time distribution, response time percentiles over time, number of requests per second, etc. These reports can help users develop quick insights into the system performance and possible bottlenecks. Our framework will also collect and download more fine-grained data including

- The RTT between the load generator and the load balancer, which is measured using the {ping} command.

- The start time and completion time of each request, which is observed on the load generator node.

- The service time of each request, which is observed on each worker node. We record the system time before and after the processing of each request, then attach the elapsed time as a filed in the json response. We will then use Gatling to parse the json responses to extract the service times, and save them into a separate file. Similarly, any information contained in the response can be parsed and saved.

- The CPU utilization of each worker node throughout the duration of the experiment. This is collected by running the {mpstat} command. Our automation script will start {mpstat} before the benchmark starts and let it run at 1 second intervals. The command output will be saved to a file and downloaded later after the benchmark completes.

These fine-grained data can be processed to generate more detailed performance metrics if needed.

Our performance benchmarking framework is released as an open-source artifact, available at https://github.com/umassos/edge-cloud-benchmarking. We have created two tags for the code versions that were used to produce the results in our paper: "sc21" for experiments with homogeneous cluster (i.e, all the VMs in the cluster are of the same instance type) and "sc21-hybrid" for experiments with heterogeneous cluster (i.e., part of the VMs in the cluster are CPU instances while the others are GPU instances).

Author-Created or Modified Artifacts:

Persistent ID: 10.5281/zenodo.5163850

```
Artifact name: Performance Benchmarking Framework for
↪  Edge/Cloud Environments
```