INVAR: Inversion Aware Resource Provisioning and Workload Scheduling for Edge Computing

Bin Wang*, David Irwin†, Prashant Shenoy*, and Don Towsley*
University of Massachusetts Amherst, USA
Email: *{binwang, shenoy, towsley}@cs.umass.edu, †irwin@ecs.umass.edu

***Abstract*—Edge computing is emerging as a complementary architecture to cloud computing to address some of its associated issues. One of the major advantages of edge computing is that edge data centers are usually much closer to users compared to traditional cloud data centers. Therefore, it is commonly believed that for developers of latency-sensitive applications, they can effectively reduce the overall end-to-end latency by simply transitioning from a cloud deployment to an edge deployment. However, as recent work has shown, the performance of an edge deployment is vulnerable to a couple of factors which under many practical scenarios can lead to edge servers providing worse end-to-end response time than cloud servers. This phenomenon is referred to as *edge performance inversion*. In this paper, we propose resource allocation and workload scheduling algorithms that actively prevent edge performance inversion. Our algorithms, named INVAR, are based on queueing theory results and optimization techniques. Evaluation results show that INVAR can find a near-optimal solution that outperforms the performance of a cloud deployment by an adjustable margin. Simulation results based on production workloads from Akamai data centers show that INVAR can outperform common heuristic-based edge deployment by 11% to 24% in real-world scenarios.**

***Index Terms*—edge computing, cloud computing, model-driven resource management.**

I. Introduction

While cloud computing has been popular for hosting a variety of online applications and services, a new class of applications has emerged in recent years that are characterized by tight latency requirements. Examples of such applications include mobile augmented and virtual reality (AR/VR), autonomous vehicle navigation and control, online gaming, and Internet of Things (IoT). The stringent latency requirements of such workloads can not be satisfied by traditional cloud servers, which are distant and incur higher latency. Edge computing has emerged as a promising solution for such applications, and involves deploying computing and storage resources at the edge of the network and close to users and end devices. Major cloud providers and even cellular telecom providers have begun to offer edge computing services to address the needs of such application workloads.

Since edge resources are closer to users and their devices, the network latency to edge servers tends to be small and much lower than the latency to cloud servers. Consequentially, conventional wisdom holds that deploying applications on edge servers (e.g., AWS Local Zones [1]), which are much closer to users than traditional cloud data centers, can effectively reduce the end-to-end latencies observed by users [2].

This work was supported in part by the National Science Foundation under awards 2211888, 2211302, and 2105494, and in part by the Army Research Lab under Cooperative Agreement W911NF-17-2-0196.

However, recent research has shown that there are practical scenarios where edge servers become a bottleneck, causing edge performance to become worse than cloud performance. Specifically, the end-to-end latency of an application is the sum of the network latency and the server processing latency. Since edge clusters are often resource-constrained, edge processing can incur high queueing delays, especially when bottlenecks arise, causing high end-to-end latency for edge servers. The work of [3] showed theoretical bounds when edge end-to-end latency can be higher than the cloud end-to-end latency, and also experimentally demonstrated such abnormal behavior using real cloud applications and workloads. This counter-intuitive phenomenon, which is referred to as *edge performance inversion*, is problematic for applications when it occurs since the higher edge latencies are worse than cloud latencies. While the work of [3] provided theoretical bounds when such performance inversion occurs and also validated the bounds experimentally, it did not propose any solutions to avoid such problems in practice.

An edge platform can employ one of two methods to avoid performance bottlenecks and high queueing delays for hosted applications. One approach is to dynamically provision more resources when queueing delays exceed a threshold, such that the additional capacity reduces the probability of an edge inversion. However, additional capacity may not always be available at a resource-constrained edge location. In this case, requests can be redirected to another nearby edge location to be serviced. While provisioning additional resources increases server costs, redirecting requests to other locations increases end-to-end latencies. Hence, both approaches need to be used judiciously to maintain good edge performance in a cost-efficient manner.

In this paper, we present INVAR: INVersion-Aware Resource provisioning and workload scheduling that is designed to avoid the edge performance inversion problem and maintain good edge performance. In designing, implementing, and evaluating INVAR, our paper makes the following contributions:

1) We define the inversion-aware resource provisioning and workload scheduling problem, and introduce a queueing theory framework to model the performance of different deployment plans.

2) We develop two optimization algorithms based on queueing theory results to determine the optimal performance that can be achieved using a given budget, and a search procedure that combines the two optimization algorithms to find the deployment plan that satisfies a given performance target with the lowest cost.
3) We implemented both numerical analysis and discrete event simulation to evaluate the proposed algorithms. Evaluation results on both synthetic scenarios and real-world workloads show that deployment plans generated by INVAR constantly outperforms cloud deployment and other edge deployments in all cases. Simulation results based on production Akamai workloads show that INVAR can outperform common heuristic-based edge deployment by 11% to 24% in real-world scenarios.

The rest of this paper is structured as follows: In section II we introduce the background of our research as well as formulate the problem statement. In section III we explain the details of the two optimization algorithms and the INVAR deployment search procedure. In section IV we show the evaluation results of INVAR using both numerical analysis and simulation. Section V introduces related work in the field. Finally, section VI concludes our work and discusses possible future work directions.

II. Background

In this section, we discuss the background on edge computing and the edge performance inversion problem, as well as our problem statement.

A. Edge Computing

In recent years, edge computing has emerged as a complementary architecture to cloud computing. It's believed to be especially suited for latency-sensitive and bandwidth-sensitive applications due to their close proximity to users and end devices. Our work assumes an *edge cloud model* of edge computing where edge resources are offered to applications using a cloud-like on-demand paradigm. Similar to conventional cloud platforms, we assume that the edge cloud is distributed across a large number of geographic locations, with each server location consisting of a small server cluster that can run applications in containers or virtual machines. This approach is similar to the cloudlets model [4] that has been advocated previously. Cloud providers have begun to offer edge cloud services by augmenting their hyper-scale cloud data centers with smaller edge data centers. In doing so, traditional cloud platforms now offer a choice of traditional cloud servers, as well as edge servers, to application developers. A developer can choose to run their application on cloud servers, edge servers, or a combination of both, depending on their needs.

Each edge cluster within the edge cloud is assumed to be deployed close to the users in order to offer a low network latency to applications. Modern edge platforms are able to provide sub ten-millisecond network latencies. For instance, Amazon Web Services (AWS) has deployed edge data centers in metropolitan areas called Local Zones [1] that can provide

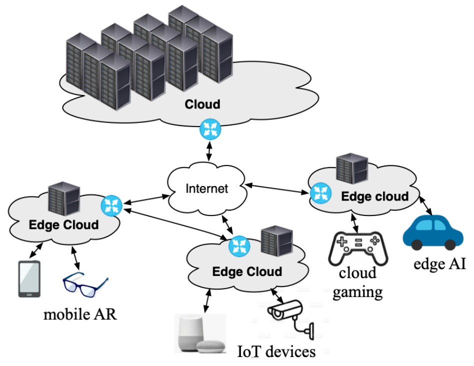


Fig. 1: Edge clouds offer lower network latency than traditional cloud platforms.

single-digit millisecond latency. In contrast, cloud platforms offer latencies of tens of milliseconds to their users. A recent measurement study of production edge and cloud platforms showed that only 3-23% of users had latencies to cloud platforms less than 10ms (depending on the cloud platform), and only 22-52% had latencies less than 20ms [5]. This large-scale study confirmed that edge network latencies are an order of magnitude lower than their corresponding cloud latencies. These lower network latencies and high bandwidth are two significant advantages of edge computing for latency-sensitive applications [6].

B. The Edge Performance Inversion Problem

As described in [3], the term edge performance inversion is used to describe the phenomenon where users experience worse end-to-end latencies with edge servers than cloud servers despite the significant network latency advantage of the edge. Below we motivate the edge performance inversion problem by discussing two possible causes of edge performance inversion.

Cause 1: The Bank teller analogy. The primary cause of edge performance inversion can be understood using the well-known Bank Teller problem [7], [8], a classical problem from queueing theory. The bank teller problem tells us that customers entering a bank always see lower waiting times when using a single queue for all tellers versus a separate queue per teller. This is because the time needed to service each customer varies from customer to customer, and in the case of separate queues, 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.

To understand how this analogy applies to edge computing, consider a cloud gaming application as a representation example of an edge application. Suppose the game service is deployed at multiple edge locations in order to serve geographically distributed users with low latency. In this case, each edge location maintains a separate queue for incoming requests from game users (as shown in Figure 2a). An alternative approach is to deploy the entire application in a single cloud

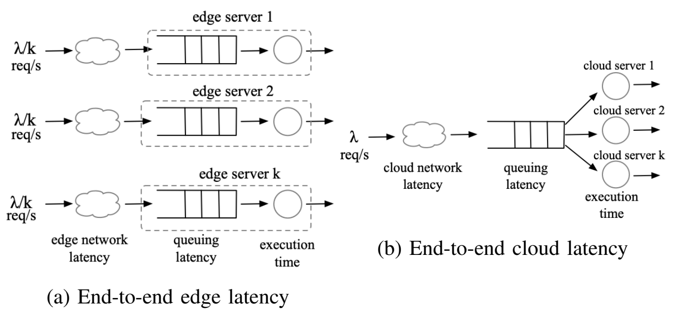


Fig. 2: End-to-end latency comparison with edge servers and cloud servers.

data center. In this case, the application uses the same number of aggregate servers as the distributed edge deployment but uses a single queue to service all requests arriving at the cloud deployment (as shown in Figure 2b). The Bank Teller problem tells us that maintaining separate queues in the edge deployment causes requests to experience higher queueing delays than a single queue in the case of the cloud deployment. Since the edge has much lower network latency than the cloud, at low utilization levels, these higher queueing delays are offset by the lower network latency, still yielding better overall end-to-end latency than the cloud. However, as utilization rises, there is a corresponding rise in queueing delays (or ”wait times”). Since edge queueing delays rise faster than the cloud ones due to the bank teller analogy, there is a cross-over point where edge performance becomes worse than the cloud. This cross-over point is depicted in Figure 3 and occurs when the higher edge queueing delays offset the benefits of lower edge network latency. Importantly, such performance inversion can occur even at server utilization levels of as low as 40% in real-world settings.

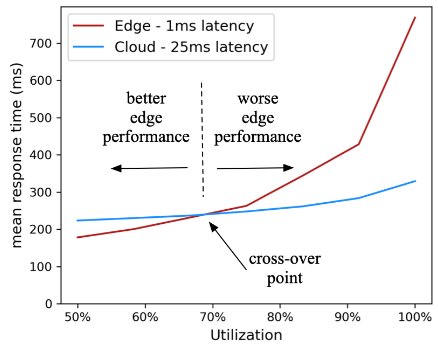


Fig. 3: Edge latency becomes worse than cloud latency above a threshold utilization.

Cause 2: Elastic Scaling Under Resource Constraints. The above bank teller example assumes a simple Poisson workload and a static number of servers for the application. While edge performance inversion can occur in such simple settings, in practice, edge and cloud applications exhibit dynamic workloads with temporal and spatial variations. Such fluctuations will further exacerbate the performance inversion problem at edge locations, making it even more likely to occur in practice. However, sophisticated applications are designed to counter the impact of dynamic workload variations through elastic scaling [9]. Elastic scaling enables an application to dynamically vary the number of servers to match the observed workload fluctuations, and thereby prevent overloads. However, edge data centers are often constrained in terms of the cluster size which limits its ability to elastically scale. In contrast, due to their massive sizes, cloud data centers rarely have resource pressure and can elastically scale applications whenever workloads rise. This is the second cause of performance inversion, where resource-constrained edge sites are limited in their ability to elastically scale overloaded applications, which causes the edge to yield worse performance than the cloud.

C. Problem Statement

We consider a system with L user locations where requests can originate from. Let $1/\gamma_i$ denote the mean inter-arrival time of requests from user location i. There are K data centers where requests can be serviced and they can be a mix of cloud data centers and edge data centers. A deployment plan needs to determine how many servers should be allocated in each data center, denoted by a server allocation vector $c = \{c_1, \ldots, c_K\}$ where c_j is the number of servers to be provisioned at data center j. c_j is also bounded by a maximum number of servers that can be allocated in data center j, denoted by C_j.

When a request arrives at data center j, it will be served by one of the idle servers, or placed in a central queue if all c_j servers are busy. Requests in the queue will have access to the next available server in a first-come-first-serve (FCFS) manner. Let $T^S_j = 1/\mu_j$ denote the mean service time for requests in data center j. We assume that the servers in the same data center are homogeneous but can be heterogeneous across data centers. Each allocated server in the data center has an associated “cost”, denoted by w_j, and the values can also be different across data centers. Depending on the context, it can be real monetary cost or other considerations (e.g., it can be the carbon intensity associated with each server for a developer who wants to minimize the carbon emission of her application). Given a server allocation vector c, the total cost would be $W(c) = \sum_{j=1}^{K} c_j w_j$.

Since both the user locations and data centers are geologically distributed, there are latencies associated with requests propagating through the network between user locations and data centers. Let $T^N_{i,j}$ denote the round-trip time (RTT) between user location i and data center j. As we have explained in the previous section, it’s not always optimal to send requests from one user location only to its nearest data center: the local data center might be overloaded, or sharing servers (with requests from other user locations) at a further away data center could lead to shorter queueing delay which outweighs the larger network latency. Therefore, we use a probabilistic scheduling approach instead: when a request from user location i arrives, one of the K data centers will be probabilistically selected, and the incoming request will be sent to the selected data center for processing. Let $p_{i,j}$ denote

TABLE I: Table of Notations

L	Number of user locations
K	Number of data centers
γ_i	Arrival rate at user location i
$p_{i,j}$	Probability that an incoming request at user location i is sent to data center j for processing
λ_j	Effective arrival rate at data center j; $\lambda_j = \sum_{i=1}^{L} \gamma_i p_{i,j}$
C_j	The maximum number of servers that can be allocated at data center j
c_j	The actual number of servers to be allocated at data center j
μ_j	The mean service rate at data center j
w_j	The cost of 1 server at data center j
$T_{i,j}^N$	Network latency between user location i and data center j (RTT)
T_j^D	The average time spent in the data center for requests arriving at data center j
T^{cloud}	The mean end-to-end response time observed by all requests under the cloud deployment

the probability that an incoming request from user location i is scheduled to data center j. The scheduling probability matrix $P = [p_{i,j}]_{L\times K}$ also needs to be determined by a deployment plan.

For a developer who is transitioning their application from a cloud deployment to an edge deployment, our goal is to devise an edge deployment plan that actively avoids edge performance inversion — the generated edge deployment plan should outperform its cloud counterpart by an adjustable margin — while using the least cost. More specifically, we assume that under a given set of parameters, the developer has an existing procedure to generate a cloud deployment plan. The mean response time attained under the cloud deployment plan is denoted by T^{cloud}. Note that T^{cloud} is the only information we need about the cloud deployment plan: knowledge about the specifics of the cloud deployment plan (such as c or P) or the procedure used for generating the cloud deployment plan is not necessary. The developer will then decide a target mean response time for the edge deployment plan $T^{target} = T^{cloud} - \delta$ where δ is adjustable. Our goal is to generate an edge deployment plan that can achieve a mean response time that's lower than T^{target} with the least cost. Furthermore, since our notion of T^{target} is very general, the techniques described in the work are also of interest to application developers without existing cloud deployment plans.

III. Optimizations on Resource Provisioning and Workload Scheduling

In this section, we first propose a system queueing model to analyze the mean end-to-end response time under a given deployment plan, then we formulate the inversion-aware resource provisioning and workload scheduling algorithms based on the proposed queueing model.

A. System Queueing Model

In this section, we show how each data center could be modeled as an $M/M/c$ queueing system and how to calculate the mean end-to-end response time using a closed-form formula. Recall that we are using a probabilistic scheduling approach, with requests from user location i being scheduled to data center j with probability $p_{i,j}$. Thus the effective arrival rate at data center j can be calculated as

$$\lambda_j = \sum_{i=1}^{L} \gamma_i p_{i,j} \tag{1}$$

We assume the request arrival processes at user locations are independent Poisson processes. Probability theory results show that 1) splitting a Poisson process with a time-independent probability generates multiple independent Poisson processes, and 2) combining independent Poisson processes results in a new Poisson process [10]. Therefore, the request arrival process at each data center is also a Poisson process with a mean arrival rate equal λ_j for data center j. Assuming request service times follow the exponential distribution, and within a data center, the requests are served in a first-come-first-serve (FCFS) order with identical service rate regardless of which user location the request originates from or which server gets the request, then each data center could be effectively modeled as an $M/M/c$ queueing system.

The performance of an $M/M/c$ queueing system has been extensively studied. For all requests scheduled to data center j, the average time spent in the data center can be calculated using the following formula:

$$T_j^D = \frac{\mathcal{C}(c_j, \lambda_j/\mu_j)}{c_j\mu_j - \lambda_j} + \frac{1}{\mu_j} \tag{2}$$

where $\mathcal{C}(c, \lambda/\mu)$ is the Erlang C function which calculates the probability that all servers are busy. The formulation of the Erlang C function is:

$$\mathcal{C}(c, \lambda/\mu) = \frac{\left(\frac{(c\rho)^c}{c!}\right)\left(\frac{1}{1-\rho}\right)}{\sum_{k=0}^{c-1}\frac{(c\rho)^k}{k!} + \left(\frac{(c\rho)^c}{c!}\right)\left(\frac{1}{1-\rho}\right)} \tag{3}$$

Equipped with equation (2), we can now use the following formula to calculate the mean end-to-end response time observed by requests from all user locations, given a deployment plan (c, P):

$$T(c, P) = \sum_{i=1}^{L} \frac{\gamma_i}{\|\gamma\|_1} \sum_{j=1}^{K} p_{i,j}(T_{i,j}^N + T_j^D) \tag{4}$$

B. Performance Optimization Given a Cost Budget

As discussed in section II-C, our goal is to generate a deployment plan that can achieve T^{target} with the minimum cost. Our first step in tackling this problem is to develop an algorithm for finding a deployment plan with the optimal performance (mean end-to-end response time) under a given budget W. This can be effectively modeled as the optimization problem below:

$$\min_{c,P} \quad T(c,P) \tag{5}$$
$$\text{s.t.} \quad 0 \le p_{i,j} \le 1 \quad \forall i \in 1 \dots L, \forall j \in 1 \dots K \tag{6}$$
$$0 \le c_j \le C_j \qquad \forall j \in 1 \dots K \tag{7}$$
$$\sum_{j=1}^{K} p_{i,j} = 1 \qquad \forall i \in 1 \dots L \tag{8}$$
$$\lambda_j < c_j \mu_j \qquad \forall j \in 1 \dots K \tag{9}$$
$$\sum_{j=1}^{K} c_j w_j \le W \tag{10}$$

Constraint (9) guarantees that no data center is overloaded (meaning that requests arrive at a faster rate than they can be serviced, which leads to unbounded queueing delay), and constraint (10) guarantees that the cost of the deployment plan is below the budget W.

However, this optimization problem cannot be solved in its original form because $T(c,p)$ requires computing the Erlang C function which is defined on discrete values of c. This makes the optimization problem a mixed-integer programming problem which is NP-hard. In addition, the computation overhead of the Erlang C function is very high, which renders the problem intractable in many cases even with an advanced MIP solver.

To address this problem, we use a continuous upper bound of the Erlang C function proposed in [11] as an approximation. The formula of the upper bound is as follows:

$$\mathcal{C}(c, \lambda/\mu) \le \left[\frac{\lambda}{c\mu} + \eta\left(\frac{\Phi(\alpha)}{\phi(\alpha)} + \frac{2}{3}\frac{1}{\sqrt{c}}\right)\right] \tag{11}$$

where $\alpha = \sqrt{-2c(1-\rho+\ln\rho)}$, $\beta = (c-\lambda/\mu)/\sqrt{\lambda/\mu}$, $\eta = (c-\lambda/\mu)/\sqrt{c}$. $\Phi(\alpha)$ is the cumulative distribution function (CDF) of the standard normal distribution variable, and $\phi(\alpha)$ is the probability density function (PDF) of the standard normal distribution variable which has the formula $\phi(\alpha) = (1/\sqrt{2\pi})e^{-(1/2)\alpha^2}$. Not only is the approximate upper bound much more efficient to compute than the original formula of the Erlang C function, it also makes c become a continuous decision variable. In addition, we can also relax constraint (10) using the Lagrangian relaxation technique [12] to make the problem easier to solve. The final formulation of the optimization is listed below:

$$\min_{c,P} \quad T'(c,P) + \tau\left(\sum_{j=1}^{K} c_j w_j - W\right)^2 \tag{12}$$
$$\text{s.t.} \quad 0 \le p_{i,j} \le 1 \quad \forall i \in 1 \dots L, \forall j \in 1 \dots K \tag{13}$$
$$0 \le c_j \le C_j \qquad \forall j \in 1 \dots K \tag{14}$$
$$\sum_{j=1}^{K} p_{i,j} = 1 \qquad \forall i \in 1 \dots L \tag{15}$$
$$\lambda_j < c_j \mu_j \qquad \forall j \in 1 \dots K \tag{16}$$

where $T'(c,P)$ is the overall mean response time formula (4) calculated with the approximate upper bound of Erlang C instead of the exact value. In the remainder of this paper, we will refer to this optimization as the **performance optimization** problem.

C. Flow Optimization Given a Server Allocation Vector

Since the solution obtained from the performance optimization problem yields a fractional solution for the server allocation vector c, it needs to be rounded to an integer vector for the final solution. However, after rounding the scheduling probability matrix P is no longer optimal. So we use a similar optimization problem to readjust P to produce the optimal mean response time, as shown below.

$$\min_{P} \quad T(c,P) \tag{17}$$
$$\text{s.t.} \quad 0 \le p_{i,j} \le 1 \quad \forall i \in 1 \dots L, \forall j \in 1 \dots K \tag{18}$$
$$\sum_{j=1}^{K} p_{i,j} = 1 \qquad \forall i \in 1 \dots L \tag{19}$$
$$\lambda_j < c_j \mu_j \qquad \forall j \in 1 \dots K \tag{20}$$

In the remainder of this paper, we will refer to this optimization as the **flow optimization** problem, as it only optimizes the scheduling probabilities which affects how requests "flow" between user locations and data centers. While the structure of the flow optimization seems similar to the performance optimization problem, there are two key differences:

1) In performance optimization the server allocation vector c is a decision variable, while in flow optimization it is an input parameter. As a result the flow optimization problem no longer needs to consider the budget constraint.
2) In performance optimization we need to use $T'(c,P)$ which is based on the approximate upper bound of Erlang C in the objective function, while in flow optimization it's viable to use $T(c,P)$ which is based on the exact Erlang C calculation since we only need to calculate the derivatives of P. Although it has higher computation complexity, the objective function now has a convex structure which means it can be solved in polynomial time.

D. Inversion-aware Deployment Plan Searching

In this section, we present an inversion-aware deployment search algorithm that combines the two optimization algorithms described in the previous sections, such that application developers can apply this algorithm to find a deployment plan that achieves a given performance target with the lowest cost. The pseudocode of this algorithm is described in Algorithm 1. Our search algorithm has two phases: first, we use the PERF_OPT function which solves the performance optimization problem to do a binary search on the budget to determine the least cost needed to satisfy the given mean end-to-end response time target. This works because when other parameters stay fixed, $T'(c, P)$ is monotonic with respect to W. In the second phase, we use a specific round function to convert the fractional solution returned by PERF_OPT into an integer solution before piping it into FLOW_OPT which solves the flow optimization problem. Combining the rounded integer server allocation vector with the scheduling probability matrix returned by FLOW_OPT would produce the final solution.

Algorithm 1 Inversion Aware Deployment Plan Search Algorithm

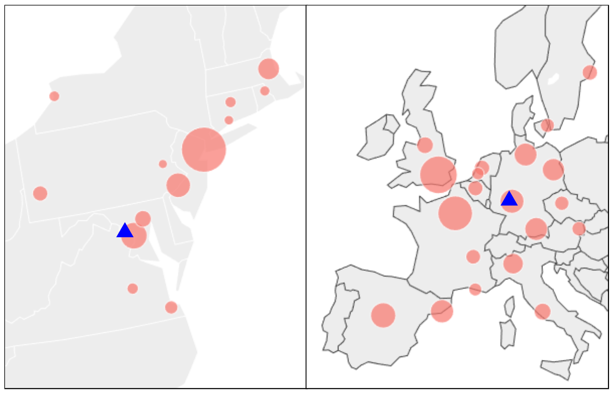
```
Input: $L$, $\gamma$, $K$, $C$, $\mu$, $w$, $T^N$, $T^{cloud}$, $\delta$, $W$, $\epsilon$
Output: $c$, $P$
 1: $T^{target} \leftarrow T^{cloud} - \delta$
 2: $W^l \leftarrow 0$
 3: $W^u \leftarrow W$
 4: while $W^u - W^l > \epsilon$ do
 5:     $W^m \leftarrow \frac{W^l + W^u}{2}$
 6:     $\text{status}, c, P \leftarrow \text{PERF\_OPT}(L, \gamma, K, C, \mu, w, W^m)$
 7:     if status = solved and $T'(c, P) \leq T^{target}$ then
 8:         $W^u \leftarrow W^m$
 9:     else
10:         $W^l \leftarrow W^m$
11:     end if
12: end while
13: $\text{status}, c, P \leftarrow \text{PERF\_OPT}(L, \gamma, K, C, \mu, w, W^u)$
14: if $T'(c, P) \leq T^{Target}$ then
15:     $c \leftarrow \text{round}(c)$
16:     $P \leftarrow \text{FLOW\_OPT}(L, \gamma, \mu, c)$
17:     return $c$, $P$
18: else
19:     return "NO SOLUTION FOUND"
20: end if
```

IV. Evaluation Results

A. Evaluation Setup

We use both numerical analysis and simulation to evaluate our algorithms. For numerical analysis, we implemented INVAR using Julia [13], a free and open-source programming language widely used for scientific computing. The optimization algorithms are implemented with JuMP [14], a modeling language for mathematical optimization embedded in Julia. The performance optimization problem described in section III-B is solved using the Artelys Knitro solver [15] with the Interior-Point/Direct algorithm [16] and the multi-start feature enabled: this is important because the objective function (12) is non-convex and the multi-start procedure helps overcome local optimality by exploring different starting points. The flow optimization problem on the other hand is a convex problem so we choose Ipopt [17], an open-source solver that also implements the Interior-Point algorithm. For simulation, we implemented a discrete event simulator (DES) using Python and SimPy [18]. Both the numerical analysis program and the simulator are open source and available at https://github.com/umassos/invar.

We conducted evaluations using two different scenarios: a synthetic scenario and a real-world scenario extracted from traces collected by the Akamai content delivery network. In the synthetic scenario, we have 1 cloud data center and 8 user locations which are equally spaced on a circle centered at the cloud data center. The Akamai traces, on the other hand, were collected in August 2013 from production Akamai data centers around the world. We extracted traces in two regions from the Akamai dataset: the United States northeast region and the Europe region, as shown in Figure 4.



(a) US Northeast (b) Europe

Fig. 4: Traces extracted from the Akamai CDN dataset. The pink circles are user locations and their size is proportional to the average request arrival rate. The blue triangles are the cloud data centers (us-east-1 and eu-central-1).

B. Deployment Plans under Fixed Cost Budgets

In this section, we use a synthetic scenario to examine the deployment plans generated by performance optimization described in III-B to get a sense of how different budgets affect the generated deployment plan.

The synthetic scenario that we use for evaluation is set as follows. There are $L = 8$ user locations spaced equally on a ring that is 30ms away from a cloud data center in the center. Each user location has the same incoming arrival rate of $\gamma = 15$. In the cloud deployment, every user location will send their requests to the cloud data center. Now let's assume that the cloud provider has deployed edge data centers for 4 of the user locations (which are also equally spaced), and the edge data centers are only 1ms away from its edge location. Note that for the other 4 user locations that do not have their own edge data center, it is still closer to one of its neighboring edge data centers compared to the cloud data center. We assume that all the data centers have the same service rate $\mu_j = 10$ and per server cost $w_j = 1$. We capped the maximum capacity at 50 at the cloud data center and at 10 for the edge data centers.

Our evaluation method is as follows. We vary the server budget from 16 to 24. Then we solve the performance optimization problem with each server budget value, then round the generated allocation vector and finally solve the flow optimization problem with the rounded allocation vector to get the final deployment plan and its corresponding mean end-to-end response time. For comparison, we also calculated the performance of the cloud deployment and a local-first edge deployment (which means each user location will always try to send its requests to the closest data center) under the same server budgets.

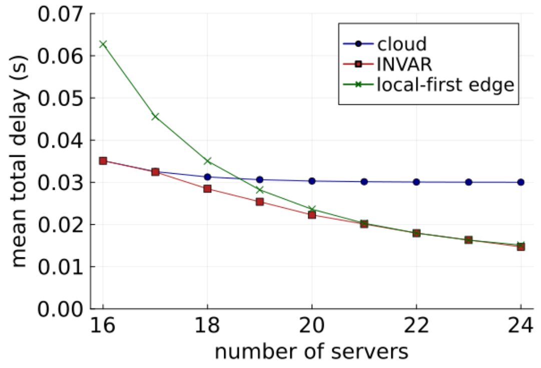


Fig. 5: Numerical analysis results of mean total delay observed with cloud deployment, local-first edge deployment, and INVAR under different server budgets.

Figure 5 shows the mean total delay (which is the sum of network latency and queueing delay) observed with each deployment technique under different budgets. We can see that with the increase of servers allocated, the delay of the cloud deployment converges the physical limit which is the network latency of 30ms. For the local-first edge deployment, it experiences the performance inversion problem (meaning that its performance is worse than a cloud deployment using the same amount of servers) when using less than 19 servers because of the long queueing delay at each data center. However, when the budget is over 19 servers, the queueing delay in the local-first deployments will become low enough for the total delay to show an advantage over the cloud deployment.

INVAR, on the other hand, outperforms the other two deployment techniques in all cases. It basically provides the same performance as the cloud deployment when the system utilization is high, and converges with the local first deployments when the system utilization becomes lower. When the system utilization (when the budget is between 18 to 20 servers) INVAR was able to balance between network latency and queueing delay by opportunistically consolidating requests from different user locations.

We can take a closer look at the topology of the deployment plans generated by INVAR in Figure 6. When the budget is 16 INVAR regresses to the cloud deployment because it's impossible to avoid performance inversion in that case, so the cloud deployment is the best we can do. When the budget is 17, INVAR was able to find a deployment plan using only 2 edge data centers out of the 4 available data centers. This is because if we deploy a few servers at a third data center, that would mean there are fewer servers at the other two data centers since the budget is fixed. In that case, although the third data center would improve the delay of the corresponding user location, the other user locations will all experience worse delays which would make the mean total delay worse. When the budget increases to 18 servers, INVAR still only uses two data centers but is able to better balance the load between the two data centers. When the budget is 19 INVAR determines there are enough resources to deploy in a third data center. Finally, INVAR switches to similar deployments as the local-first approach when the budget is 20 or more.

We also run all the generated deployments in our simulator. Our simulation results closely match the numerical analysis results, as shown in Figure 7.

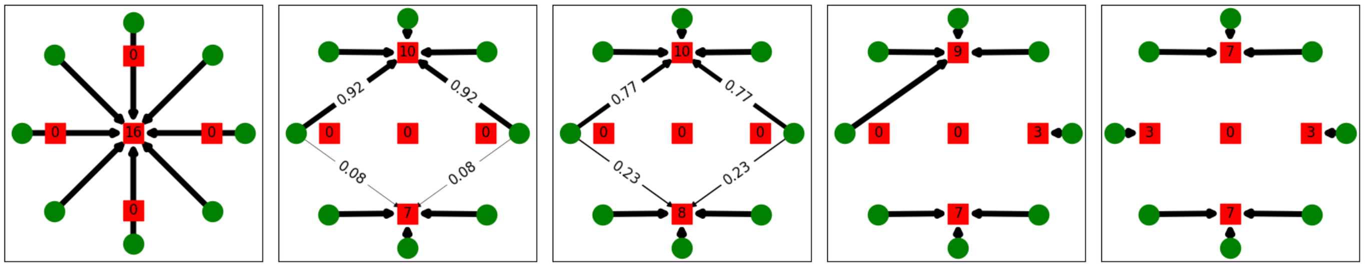
C. Akamai Workload Evaluation

In this section, we evaluate the effectiveness of the inversion-aware deployment search algorithm described in section III-D. For this evaluation, we are using traces collected by the Akamai content delivery network in August 2013. The original traces contain the amount of data served by Akamai data centers around the world. We extracted two regions: the US northeast region (13 cities) and the Europe region (20 cities), as shown in Figure 4. We assumed the cloud data center is located in the AWS us-east-1 (N. Virginia) region for the US northeast scenario and the AWS eu-central-1 (Frankfurt) region for the Europe scenario. We also assumed that the edge data centers are deployed in available or announced AWS Local Zone locations. There are some key differences between these two regions:

1) The requests distribution is very skewed in the US northeast scenario: the top-4 cities generate more than 80% of the total requests, while in the Europe scenario, the request distribution is more balanced.
2) The cloud latency in general is smaller in the US northeast scenario than the Europe scenario: the largest cloud latency in the US northeast scenario is less than 20ms, while in the Europe scenario, it's over 40ms.
3) There are more data centers in the Europe scenario than in the US northeast scenario. There are over 10 AWS data centers (regions plus Local Zones) in the Europe scenario but only 4 AWS data centers in the US northeast scenario.

For both scenarios, first, we run a cloud deployment with 60% utilization to get a baseline performance. Next, we run a local-first heuristic edge deployment that deploys servers at the nearest data center to match the cloud utilization. Finally, we run INVAR to generate deployment plans. We picked different δ values for the two scenarios: since the cloud is generally further away and there are more data centers available in the Europe region, this means it's possible to achieve a higher latency cut in the Europe scenario. Therefore, we picked $\delta = 0.005$ (5ms) for the US northeast scenario and $\delta = 0.010$ for the Europe scenario.

The response times observed under different deployments are shown in Figure 8. We can see that for the local-first edge deployments, the observed response time is very volatile and performance inversion happened in both cases. Also, notice that the local-first approach yields worse performance



(a) budget=16 (b) budget=17 (c) budget=18 (d) budget=19 (e) budget=20

Fig. 6: Topology of deployment plans generated by INVAR under different budgets. The green circles represent user locations, and the red squares represent data centers (1 cloud data center in the middle and 4 edge data centers near 4 of the user locations). The number on each data center represents the number of servers to be allocated at that data center. The arrows represent how the requests are scheduled from user locations to data centers, and the number on each arrow is the corresponding scheduling probability if it's not 1.

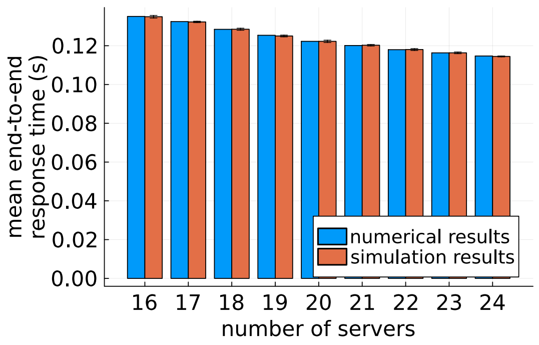


Fig. 7: Numerical analysis and simulation results of mean end-to-end response time observed with INVAR under different server budget.

in the Europe scenario than the US northeast scenario. This is because the European region has more user locations, more evenly distributed requests, and more data centers, which is equivalent to more queues in the Bank Teller analogy which accounts for higher queueing delay.

On the other hand, INVAR was able to avoid edge performance inversion in all cases, while constantly achieving the response time target with respect to δ. On average INVAR outperforms the local-first edge deployments by 23.9% in the Europe scenario and by 10.7% in the US northeast scenario. This does come at a cost of using more servers than the cloud deployment and the local-first deployment, as shown in Figure 9, but we argue that the extra cost is necessary in order to fulfill the promise of lower latency of edge computing. Figure 10 shows a more detailed graph of how INVAR provisions servers across different data centers. We can see that although there are 4 available data centers, INVAR only deploys in 3 data centers, and only in 2 data centers when the load is low in the early morning. The Philadelphia data is left empty except for peak hours: this is because INVAR would usually send requests from Philadelphia to the New York data center instead since the New York data center is very close to Philadelphia. By doing so INVAR reduces the queueing delay seen by requests from Philadelphia which offsets the slightly longer network latency, while also reducing the overall server cost.

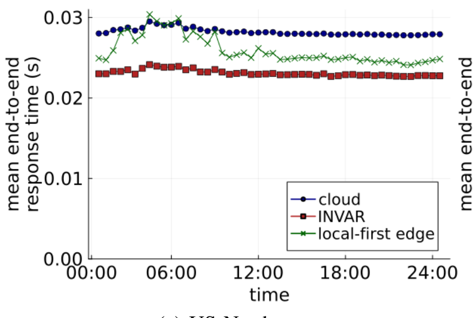
D. Sensitivity Analysis

In our model, we assume both the inter-arrival times and service are exponentially distributed, while in reality, this may not always be the case. In this section, we conduct sensitivity analysis using simulation to study how different inter-arrival time and service time distributions affect the accuracy of INVAR. We do this using the Akamai traces in the US Northeast region, and we use the deployment plans generated in Section IV-C. We run the traces and deployment plans in our simulator but with different distributions plugged in for the inter-arrival time generation at user locations and service time generation at data centers. We compared the original numerical analysis and simulation results from Section IV-C with three alternative distributions: uniform distributions, gamma distribution with shape parameter 3, and gamma distribution with shape parameter 5. All three alternative distributions still have the same mean as the original exponential distributions.

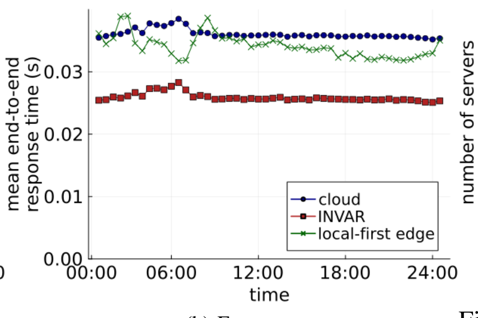
The simulated mean end-to-end response times are reported in Tables II and III, together with the numerical analysis and simulation results using the original exponential distributions. We can observe that alternative distributions do change the response time result, but the effect is not very significant (the difference is within 2% range for all cases). Moreover, in all the cases tested the mean response times with alternative distributions are actually lower than the original results, which means our deployment plans are still sufficient to yield better response time than the performance target.

TABLE II: Mean end-to-end results using different inter-arrival time distribution in the US Northeast scenario

Method — Distribution	mean	95% CI
Numerical Analysis — $E(\gamma)$	0.0229955	–
Simulation — $E(\gamma)$	0.0230106	(0.0229797, 0.0230415)
Simulation — $U(0, 2/\gamma)$	0.0227394	(0.0227214, 0.0227574)
Simulation — $\Gamma(3, 3\gamma)$	0.0226909	(0.022672, 0.0227097)
Simulation — $\Gamma(5, 5\gamma)$	0.0226576	(0.0226449, 0.0226702)



(a) US Northeast



(b) Europe

Fig. 8: Mean end-to-end response time using different deployments

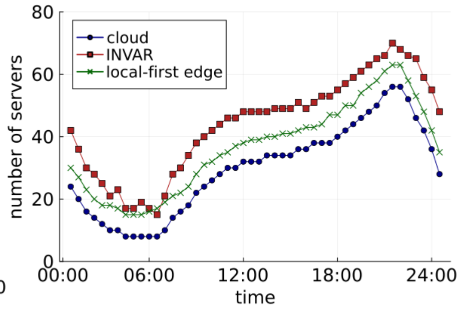


Fig. 9: Total cost of different deployments in the Europe region.

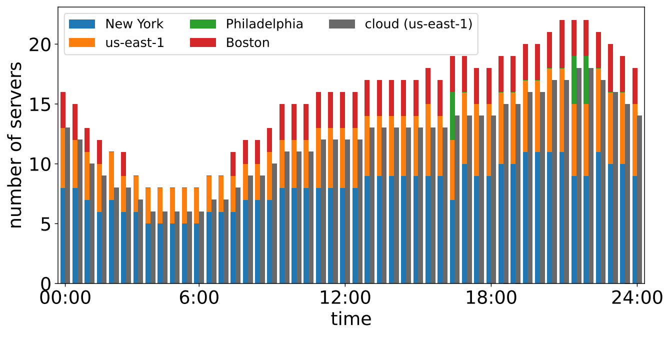


Fig. 10: Server allocation using INVAR versus cloud deployment

TABLE III: Mean end-to-end results using different service time distribution in the US Northeast scenario

Method — Distribution	mean	95% CI
Numerical Analysis — $E(\mu)$	0.0229955	–
Simulation — $E(\mu)$	0.0230106	(0.0229797, 0.0230415)
Simulation — $U(0, 2/\mu)$	0.0228921	(0.0228825, 0.0229017)
Simulation — $\Gamma(3, 3\mu)$	0.0228781	(0.022865, 0.0228911)
Simulation — $\Gamma(5, 5\mu)$	0.0228574	(0.0228469, 0.0228679)

V. Related Work

Model-driven resource allocation: Queueing theory models have recently been widely used for resource allocation problems. Examples include optimal multi-core chip design with constrained power or area budget [19], resource allocation in multimedia cloud to provide services with minimal response time or minimal cost [20], container allocation for serverless functions with tail latency SLA requirements [21], and tenant placement and resource provisioning in multi-tenant SaaS [22]. However, to our knowledge, no previous work has been proposed to address the performance inversion problem in edge computing. In this work, we use queueing theory models to avoid the edge performance inversion problem while considering performance-cost tradeoffs.

Resource scheduling in Edge Computing: Resource scheduling in edge computing has been widely studied [6]. [23] proposes a resource allocation mechanism for managing computational and communication resources for AR applications at the edge. [24] proposes an optimization algorithm for resource allocation and computation offloading in vehicular edge networks. [25] investigates resource allocation and task placement for IoT applications. Our approach doesn't make many assumptions about the workload, and our model is general enough to be applied in many different scenarios. Moreover, much of this work focuses on optimizing server utilization or energy consumption, while the vulnerability of edge performance is often overlooked. We argue that the overall end-to-end latency needs to be prioritized because low latency is a major selling point for edge computing adoption.

VI. Conclusions and Future Work

In this work, we introduce INVAR, an algorithm based on queueing theory results and optimization techniques for provisioning edge deployments that are guaranteed to avoid edge performance inversion. Our algorithm considers the workload distribution, the user location and data center topology, and the edge data center resource constraints, based on which INVAR will search the parameter space to find a deployment plan that can achieve a given performance target with the least cost. We have validated INVAR using both numerical analysis and simulation. Simulation results using traces extracted from Akamai production workloads show that INVAR-generated deployments can avoid performance inversion in real-world scenarios while outperforming a common heuristic-based edge deployment by 11% to 24%.

There are several potential directions for future work. When a request arrives at a data center, in some cases it needs to be processed by more than one server or container, e.g., in microservices and serverless function composition. In that case, the data center may not be suitable to be modeled as an $M/M/c$ queueing system but rather a directed acyclic graph (DAG) of queueing networks, which requires certain adjustments to our model. Another direction is that right now, our model requires the budget W and performance difference δ as user inputs. As these parameters are crucial for the deployment plan search, it could be tricky for users to figure out the right values. A mechanism that can handle those parameters automatically would be a useful improvement.

References

[1] "AWS Local Zones," https://aws.amazon.com/about-aws/global-infrastructure/localzones/.

[2] B. Varghese, E. de Lara, A. Y. Ding, C.-H. Hong, F. Bonomi, S. Dustdar, P. Harvey, P. Hewkin, W. Shi, M. Thiele, and P. Willis, "Revisiting the Arguments for Edge Computing Research," *IEEE Internet Computing*, vol. 25, no. 5, pp. 36–42, Sep. 2021.

[3] A. Ali-Eldin, B. Wang, and P. Shenoy, "The hidden cost of the edge: A performance comparison of edge and cloud latencies," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, Nov. 2021, pp. 1–12.

[4] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[5] B. Charyyev, E. Arslan, and M. H. Gunes, "Latency Comparison of Cloud Datacenters and Edge Servers," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, Dec. 2020, pp. 1–6.

[6] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, "Resource Scheduling in Edge Computing: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2131–2165, 2021.

[7] J. F. C. Kingman, "Inequalities in the Theory of Queues," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 32, no. 1, pp. 102–110, 1970.

[8] J. J. Buckley, *Simulating Fuzzy Systems*. Springer Science & Business Media, Feb. 2005.

[9] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "AGILE: Elastic distributed resource scaling for Infrastructure-as-a-Service," in *10th International Conference on Autonomic Computing (ICAC 13)*. San Jose, CA: USENIX Association, Jun. 2013, pp. 69–82.

[10] H. Pishro-Nik, "Introduction to Probability, Statistics and Random Processes," *Electrical and Computer Engineering Educational Materials*, Jan. 2014.

[11] A. J. E. M. Janssen, J. S. H. van Leeuwaarden, and B. Zwart, "Refining Square-Root Safety Staffing by Expanding Erlang C," *Operations Research*, vol. 59, no. 6, pp. 1512–1522, Dec. 2011.

[12] M. L. Fisher, "The Lagrangian Relaxation Method for Solving Integer Programming Problems," *Management Science*, vol. 50, no. 12_supplement, pp. 1861–1871, Dec. 2004.

[13] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A Fresh Approach to Numerical Computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, Jan. 2017.

[14] M. Lubin, O. Dowson, J. D. Garcia, J. Huchette, B. Legat, and J. P. Vielma, "JuMP 1.0: Recent improvements to a modeling language for mathematical optimization," *Mathematical Programming Computation*, Jun. 2023.

[15] R. H. Byrd, J. Nocedal, and R. A. Waltz, "Knitro: An Integrated Package for Nonlinear Optimization," in *Large-Scale Nonlinear Optimization*, ser. Nonconvex Optimization and Its Applications, G. Di Pillo and M. Roma, Eds. Boston, MA: Springer US, 2006, pp. 35–59.

[16] R. Waltz, J. Morales, J. Nocedal, and D. Orban, "An interior algorithm for nonlinear optimization that combines line search and trust region steps," *Mathematical Programming*, vol. 107, no. 3, pp. 391–408, Jul. 2006.

[17] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, Mar. 2006.

[18] "SimPy 4.0.1 documentation," https://simpy.readthedocs.io/en/4.0.1/.

[19] C. Delimitrou and C. Kozyrakis, "Amdahl's law for tail latency," *Communications of the ACM*, vol. 61, no. 8, pp. 65–72, Jul. 2018.

[20] X. Nan, Y. He, and L. Guan, "Optimal resource allocation for multimedia cloud based on queuing model," in *2011 IEEE 13th International Workshop on Multimedia Signal Processing*, Oct. 2011, pp. 1–6.

[21] B. Wang, A. Ali-Eldin, and P. Shenoy, "LaSS: Running Latency Sensitive Serverless Computations at the Edge," in *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '21. New York, NY, USA: Association for Computing Machinery, Jun. 2021, pp. 239–251.

[22] W. Su, J. Hu, C. Lin, and S. Shen, "SLA-Aware Tenant Placement and Dynamic Resource Provision in SaaS," in *2015 IEEE International Conference on Web Services*, Jun. 2015, pp. 615–622.

[23] A. Al-Shuwaili and O. Simeone, "Energy-Efficient Resource Allocation for Mobile Edge Computing-Based Augmented Reality Applications," *IEEE Wireless Communications Letters*, vol. 6, no. 3, pp. 398–401, Jun. 2017.

[24] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint Offloading and Resource Allocation in Vehicular Edge Computing and Networks," in *2018 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2018, pp. 1–7.

[25] X. Xu, C. He, Z. Xu, L. Qi, S. Wan, and M. Z. A. Bhuiyan, "Joint Optimization of Offloading Utility and Privacy for Edge Computing Enabled IoT," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 2622–2629, Apr. 2020.