Scalable Edge Computing for Low Latency Data Dissemination in Topic-based Publish/Subscribe

Shweta Khare*, Hongyang Sun*, Kaiwen Zhang†, Julien Gascon-Samson‡,
Aniruddha Gokhale*, Xenofon Koutsoukos* and Hamzah Abdelaziz*
*Vanderbilt University, USA; Email: {shweta.p.khare, hongyang.sun, a.gokhale,
xenofon.koutsoukos, hamzah.abdelaziz}@vanderbilt.edu
†École de technologie supérieure, Montreal, Canada; Email: kaiwen.zhang@etsmtl.ca
‡University of British Columbia, Vancouver, Canada; Email: julien.gascon-samson@ece.ubc.ca

***Abstract*—Advances in Internet of Things (IoT) give rise to a variety of latency-sensitive, closed-loop applications that reside at the edge. These applications often involve a large number of sensors that generate volumes of data, which must be processed and disseminated in real-time to potentially a large number of entities for actuation, thereby forming a closed-loop, publish-process-subscribe system. To meet the response time requirements of such applications, this paper presents techniques to realize a scalable, fog/edge-based broker architecture that balances data publication and processing loads for topic-based, publish-process-subscribe systems operating at the edge, and assures the Quality-of-Service (QoS), specified as the 90th percentile latency, on a per-topic basis. The key contributions include: (a) a sensitivity analysis to understand the impact of features such as publishing rate, number of subscribers, per-sample processing interval and background load on a topic's performance; (b) a latency prediction model for a set of co-located topics, which is then used for the latency-aware placement of topics on brokers; and (c) an optimization problem formulation for k-topic co-location to minimize the number of brokers while meeting each topic's QoS requirement. Here, k denotes the maximum number of topics that can be placed on a broker. We show that the problem is NP-hard for $k \geq 3$ and present three load balancing heuristics. Empirical results are presented to validate the latency prediction model and to evaluate the performance of the proposed heuristics.**

***Index Terms*—Fog/Edge computing; Topic-based publish/subscribe; Brokers; Real-time; Scalability**

I. Introduction

The Internet of Things (IoT) is a paradigm in which a plethora of devices across many domains are interconnected to provide and exchange data. Over the last few years, the IoT landscape has grown tremendously, with some studies estimating the number of connected devices to be in the range of tens of billions [24]. In many IoT applications, large amounts of data are produced by sensors deployed at scale, and rapidly consumed in order to provide fast decision-making. This is notably the case in many Smart City applications [46] in which data acquired from many sensors must be rapidly processed in an online/streaming manner to provide low-latency results for closed-loop actuation.

For example, SmartDriver [18] is a real-time personalized recommendation service for improving driving efficiency and road safety that is deployed in the city of Seville in Spain. Here, each driver sends their biometric information, such as heart-rate and vehicle information, every 10 seconds to the SmartDriver server. SmartDriver uses this information to monitor each driver's stress level and to provide personalized recommendations for safe and fuel-efficient driving. In addition to sending their own vehicle information, drivers also subscribe to receive information about all nearby drivers (within 100 meters) to assess traffic conditions. Given the time and location sensitivity of the provided information, and the prospect of low-latency 5G networks [33], IoT applications such as SmartDriver often impose strict response time requirements [27]. Further, in a medium-sized or a large city, there can be tens of thousands of commuting vehicles at rush hour. Therefore, a scalable and low-latency solution for both data dissemination and in-network processing is needed for IoT applications.

The Publish/Subscribe (pub/sub) [16] communication pattern is considered highly suitable for the data dissemination needs of IoT applications [26], since it provides scalable, asynchronous and anonymous many-to-many communication between data producers (publishers) and data consumers (subscribers). In pub/sub systems, subscribers specify their interests in receiving data in the form of subscriptions. Publishers transmit publication messages that are disseminated by the underlying system to the relevant subscribers. The literature distinguishes between many flavors of pub/sub; the most prevalent being topic-based pub/sub, which is the subject of this paper. In topic-based pub/sub, subscriptions are expressed over *topics*, which can be used to model communication channels. Topics are specified by topic names, subscribers declare their subscriptions to specific topic names, and publishers tag their messages with topic names. The topic-based pub/sub system dispatches all the messages published on a topic to all interested subscribers.

In addition to data dissemination, IoT applications also require real-time processing of streaming data. Traditionally, data produced at the network edge is sent to the cloud for processing. However, this approach can consume very high bandwidth and incur unpredictable and large latencies. Therefore, cloud-based processing and dissemination is not the best choice for latency-critical IoT applications [47]. Recently, edge [20], [40], [41], fog [8] and mobile-cloud computing [17] models have been proposed to address these concerns and support execution of computations near the source of data

978-1-5386-9445-9/18/$31.00 ©2018 IEEE
DOI 10.1109/SEC.2018.00023

on low-cost edge devices and small-scale datacenters called cloudlets [41].

Edge computing, combined with the pub/sub communication model, which together is referred to as the *publish-process-subscribe* [31] paradigm, provides a promising approach for enabling low-latency data distribution and processing for IoT systems. In this model, computations take place on published streams of data directly at the pub/sub brokers deployed near the edge. This approach has several advantages: (1) results can be disseminated to local subscribers quickly; (2) only aggregated results are sent to the cloud backend to reduce bandwidth consumption; and (3) data can be anonymized before being sent to the cloud for privacy.

Although pub/sub brokers that perform streaming analytics are increasingly being used to enable latency-critical IoT applications, existing solutions seldom provide any Quality-of-Service (QoS) assurance on latencies experienced by the system. Providing some measure of response time assurance is imperative for the practical utility of many IoT applications. To address these concerns, in this paper, we present a solution to provide QoS specified as the desired per-topic 90th percentile latency for a publish-process-subscribe system. Per-topic 90th percentile latency QoS implies that 90% of the messages received by all subscribers for a topic will have latencies below the specified QoS value. To ensure more reliable system performance, we use QoS specified as the 90th percentile latency as opposed to average latency [1]. Our solution first learns a latency prediction model of the publish-process-subscribe broker, and subsequently uses the learned model to determine the number of edge-based pub/sub brokers needed, as well as the placement of topics on these brokers, so that the QoS requirement is met, while making efficient use of the constrained system resources.

In this regard, this paper makes the following key contributions:

- **Sensitivity analysis:** We present a sensitivity analysis of the impact of different pub/sub features including number of subscribers, number of publishers, publishing rate and per-sample processing interval, on a topic's 90th percentile latency, both in an isolated case where no other topics are hosted at the broker and in a co-located case where other topics are simultaneously hosted at the broker.
- **Latency prediction model:** We present a model for predicting a topic's 90th percentile latency based on its publishing rate, per-sample processing interval, as well as a characterization of the background load imposed by other co-located topics on a broker. Neural network regression is used to learn a separate model for hosting a different number of co-located topics on a broker up to a maximum *degree of co-location* k. The learned models are demonstrated to have $\sim$97% accuracy and experimental results show that only up to $\sim$10% of the messages in the system are not able to meet the desired latency QoS as a result of prediction error and subsequent incorrect topic placement.
- **Topic co-location heuristics:** We formulate a *k-Topic Co-location Problem* (*k*-TCP) of finding a resource-efficient co-location scheme for a collection of topics on brokers such that their desired QoS in terms of 90th percentile latency is not violated. Here, the degree of co-location k specifies the maximum number of topics that can be hosted by any broker. We show that *k*-TCP is NP-hard for $k \geq 3$ and present three heuristics that use the latency prediction model for placement of topics on brokers. The performance of these heuristics is evaluated and compared through extensive experiments.

The rest of this paper is organized as follows: Section II presents related work and compares our solution to some existing pub/sub systems. Section III gives a formal statement of the problem we are studying. Section IV shows the results of a sensitivity analysis and the learned latency prediction model. Section V presents the complexity analysis of the topic co-location problem and the proposed heuristic-based solutions. Section VI presents experimental results to validate our solutions. Finally, Section VII offers concluding remarks and describes future work.

II. Related Work

Based on the expressiveness of subscriptions supported, a pub/sub system can be: (1) Content-based [5], where subscribers specify arbitrary boolean functions on the content of the messages; (2) Attribute-based [32], where subscribers specify predicates over attribute values associated with the messages; or (3) Topic-based [22], where messages are tagged with a topic name and subscribers that are interested in a specific topic receive all messages associated with that topic.

Matching published data with subscriptions for data dissemination occurs over an overlay network of pub/sub brokers. Brokers in a pub/sub system may be organized into a tree-based overlay [12], cluster-based overlay [10], structured/unstructured peer-to-peer overlay [3], [44] or cloud-based overlay [22], [32]. Tree-based, cluster-based and peer-to-peer overlays incur multi-hop routing latencies, lack reconfiguration flexiblity and require maintenance of costly state information. Increasingly, single-hop, topic-based pub/sub systems, such as MQTT (http://mqtt.org/), ActiveMQ (http://activemq.apache.org/), Amazon IoT (https://aws.amazon.com/iot/), are being used for developing IoT applications. These systems comprise a single flat layer of pub/sub brokers that are generally deployed in the cloud. Therefore, in this paper we focus on topic-based, single-hop, pub/sub systems similar to MQTT.

Many well-known and commercially-available, topic-based pub/sub systems, such as MQTT, Redis (https://redis.io/), Kafka (https://kafka.apache.org/) and ActiveMQ have been used to build IoT applications. For example, the SmartSantander [39] IoT testbed uses ActiveMQ to distribute a variety of sensor data. MQTT is used to create a smart parking application [30] and the SmartDriver application described previously uses Kafka.

Very few pub/sub systems provide QoS guarantees [6], [11] on latency, which is much desirable for supporting latency

critical IoT applications. IndiQoS [11] reserves network level resources over a peer-to-peer overlay of brokers to ensure QoS of data delivery. However, it is not always practical to make network-level resource reservations. Harmony [45] is a peer-to-peer pub/sub system which continuously monitors link quality and adapts routing paths for low-latency data dissemination. Harmony can also make use of priority-based scheduling of messages if the underlying network supports it. DCRD [25] dynamically switches among next-hop downstream nodes for reliable and time-bound data delivery. Brokers in DCRD maintain a sorted list of next-hop nodes for each subscriber on the basis of expected delay and reliability of delivery via the next-hop node. Although these solutions support QoS for latency of data delivery, they are designed for peer-to-peer, multi-hop networks and are not directly applicable for single-hop, topic-based pub/sub systems like MQTT, ActiveMQ, etc. Moreover, these solutions primarily focus on re-routing paths for data delivery in response to changes in network link characteristics. They do not consider the impact of existing broker load on latency.

With the adoption of edge computing concepts of processing near the source of data, many pub/sub sytems have emerged that implement the publish-process-subscribe [13], [31] pattern and additionally support computation at the pub/sub brokers. Latencies in publish-process-subscribe systems will be affected significantly by processing delays at the broker in addition to network link characteristics. Therefore, managing the load at pub/sub brokers is important for ensuring acceptable performance. Typically, load in topic-based pub/sub systems is managed by placing the topics on multiple brokers and distributing the connected endpoints across these brokers. Kafka supports manual rebalancing of topic load, while Dynamoth [22] performs this rebalancing dynamically when the empirically set network thresholds are exceeded. However, both Kafka and Dynamoth do not perform load-balancing in a latency aware manner for QoS assurance. MultiPub [23] finds an optimal placement of topics across geographically distributed datacenters for ensuring per-topic 90th percentile latency of data delivery, but it only considers inter-datacenter network latencies and assumes that each datacenter has a local load balancing algorithm. On the contrary, FogMq [2] uses a distributed flocking algorithm to migrate the entire pub/sub broker between edge sites to ensure bounded tail latency of computation.

To addresss the need to balance loads at publish-process-subscribe brokers for providing latency QoS of data delivery, our solution learns a latency model for pub/sub broker load and uses the learned model for distributing the topic load across brokers such that data delivery QoS is provided in a resource efficient manner.

We use a data-driven approach instead of closed-form, analytical solutions, to model the impact of broker load on a topic's latency. Closed-form, analytical solutions, such as queueing models have been used extensively for performance modeling [4] [9] [36]. However, we use a data-driven approach since simple queueing models do not incorporate the impact of interference by other co-located topics on a topic's latency and typically assume Poisson arrivals. In IoT deployments, it is more likely that sensors publish information at constant rate. Practical use of queueing models requires us to explicitly measure the processing capacity of the broker per topic. Although it can be indirectly estimated by measuring the number of queued samples per topic, many commercial off-the-shelf pub/sub libraries do not expose this metric. To the best of our knowledge, a machine-learning based approach for modeling the performance of publish-process-subscribe systems has not been presented before.

III. Problem Statement

In this section, we first describe a use case to motivate the need for latency-bounded, edge-based publish-process-subscribe systems (Section III-A). We then present the system model (Section III-B) and assumptions made (Section III-C). Finally, we provide the formal statement of k-Topic Co-location (k-TCP) optimization problem that meets the QoS requirements while making efficient use of the broker resources (Section III-D).

A. Motivational Use Case

We use the DEBS Grand Challenge dataset [28] on New York taxi trips as our motivational use case — a near real-time, city-wide taxi navigation and dispatch service. The service divides New York into 500m$\times$500m regions and taxis within each region send their location updates on its region's *gps* topic. Additionally, taxis also subscribe to its region's *update* topic to receive processed information such as most profitable regions of operation, traffic and dispatch information. All topics are hosted by a publish-process-subscribe system running on brokers near the edge, called *edge brokers*, inside a small-scale datacenter. The RIoTBench paper [43] has benchmarked some stream processing pipelines built for the New York taxi dataset, such as ETL (Extract Transform Load), prediction, model training and statistical aggregation. While ETL and prediction pipelines take 10-40ms, statistical summarization and model training take $\sim$50 seconds. Model training and statistical summarization are good examples of latency-insensitive processing that can be offloaded to a more resourceful cloud backend, while ETL and prediction can be performed at the edge brokers to provide low-latency inference.

Given the time-sensitive nature of GPS position, traffic and dispatching information [19], we consider the response time requirement for the application to be sub-second, i.e., pre-processing of data on the *gps* topic should happen within one second and the updates published on the *update* topic should also be disseminated to all taxis within one second.

B. System Model and Notations

We now introduce the system model and notation used in the paper. Consider a system where the cloud provider operates a set of homogeneous server brokers that are deployed on fog/edge resources. Let $T = \{t_1, t_2, \cdots, t_n\}$ be a collection of

n pub/sub topics that need to be allocated on the brokers. Each topic $t_i \in T$ is characterized by several parameters, including the number of publishers, overall publishing rate, per-sample processing interval in the broker, number of subscribers, etc. Since each topic may only occupy a fraction of resources in a broker – the amount of which will be determined by a combination of its parameters described above – multiple topics can be co-located on the same broker for better resource utilization. Co-located topics, however, affect each other's performance [15], [34], thus increasing their end-to-end delays and hence 90th percentile latencies. In this paper, we allow a maximum of k topics to be co-located, where $k \geq 1$ is a constant parameter that represents the *degree of co-location*. The value of k can be determined empirically by examining the overhead of managing multiple co-located topics as well as the severity of interference in terms of the latency degradation.

Let τ denote the desired 90th percentile latency that should not be exceeded by all topics. Given τ and k, to solve the proposed k-Topic Co-location (k-TCP) problem, we consider the following two sub-problems:

(1) Design a *latency prediction model* for the 90th percentile latencies of up to k co-located topics based on their input parameters.
(2) Find a *topic co-location scheme* to minimize the number of brokers used, which is needed due to the resource-constrained nature of the edge yet ensuring that all topics satisfy the desired 90th percentile latency τ.

C. Assumptions

Our system model makes the following assumptions:

(1) We only consider the impact of a broker load's on a topic's latency. In practice, a topic's latency will also be influenced by the fluctuating network conditions. We assume constant network latency and bandwidth.
(2) For simplicity of discourse, we assume that all topics have the same latency QoS requirement τ, although our system can support differentiated per-topic QoS requirements.
(3) We assume that the per-sample processing performed at the broker is CPU-bound.
(4) We assume that all edge brokers are homogeneous, i.e., they have the same hardware specification.

D. k-Topic Co-location Problem (k-TCP)

We present a formal definition of the topic co-location problem. For a collection $T = \{t_1, t_2, \ldots, t_n\}$ of n topics, a degree of co-location k, and a latency bound τ, a topic *co-location scheme* $\mathcal{S} : T \rightarrow B$ assigns the topics to a set $B = \{b_1, b_2, \ldots\}$ of edge brokers. The goal is to minimize the number $|B|$ of edge brokers used while ensuring that each topic satisfies the desired latency τ.

Under a particular co-location scheme, let y_j to be a binary variable that indicates whether broker b_j is used, i.e.,

$$y_j = \begin{cases} 1 & \text{if broker } b_j \text{ is used} \\ 0 & \text{otherwise} \end{cases}$$

and let x_{ij} be a binary variable that indicates the assignments of topics to brokers, i.e.,

$$x_{ij} = \begin{cases} 1 & \text{if topic } t_i \text{ is assigned to broker } b_j \\ 0 & \text{otherwise} \end{cases}$$

Also, for each topic $t_i \in T$, let $T_i \subseteq T$ denote its set of co-located topics (including t_i itself) on the same broker, i.e., $T_i = \{t_{i'} \in T | x_{ij} = x_{i'j} = 1\}$, and let $\ell_i(T_i)$ denote its 90th percentile latency, which can be computed by the latency predictive model (Section IV). If topic t_i is assigned to a server alone without other co-located topics, we simply use ℓ_i to denote its 90th percentile latency. The following describes a natural property on the latency model.

Property 1. *(a) $\ell_i \leq \tau$ for all $t_i \in T$; (b) $\ell_i(T_i^{''}) \leq \ell_i(T_i^{'})$ if $T_i^{''} \subseteq T_i^{'}$ for all $t_i \in T$.*

In particular, the property states that: (a) each topic always satisfies the latency requirement when assigned alone to a broker[1]; and (b) removing a topic from a set of co-located topics on a broker will not increase the latency for any of the remaining topics.

Now, we formulate the **k-Topic Co-location Problem (k-TCP)** as the following integer linear program (ILP):

$$\text{Minimize } |B| = \sum_j y_j$$

$$\text{Subject to } \sum_j x_{ij} = 1, \quad \forall t_i \in T \tag{1}$$

$$\sum_i x_{ij} \leq k, \quad \forall b_j \in B \tag{2}$$

$$\ell_i(T_i) \leq \tau, \quad \forall t_i \in T \tag{3}$$

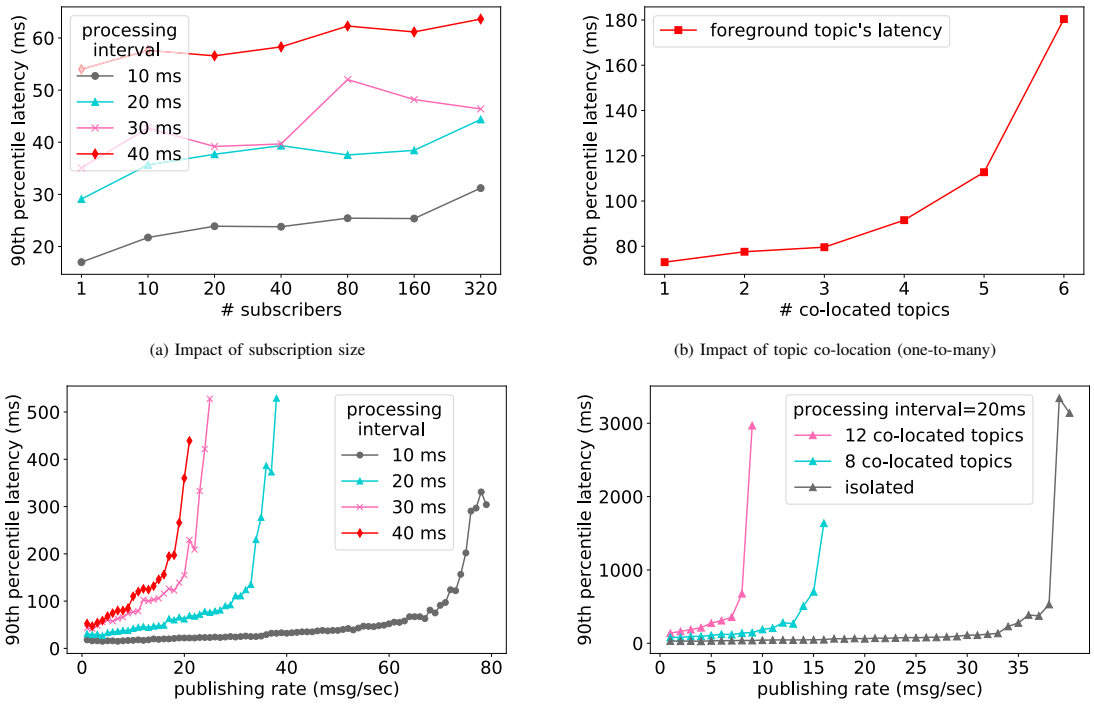
$$x_{ij}, y_j \in \{0, 1\}, \quad \forall t_i \in T, \forall b_j \in B \tag{4}$$

In the above formulation, Constraint (1) requires each topic to be assigned to exactly one broker, Constraint (2) allows no more than k co-located topics on each broker, Constraint (3) ensures the latency satisfiability for all topics, and Constraint (4) requires the decision variables to be binary. Section V shows the complexity of k-TCP and presents several heuristic solutions.

IV. Latency Prediction Model and its Sensitivity Analysis

Solutions to the proposed k-TCP problem rely on an accurate understanding of the 90th percentile end-to-end latency values per topic under different topic co-location scenarios. To that end, we build a latency prediction model to determine the latency satisfiability of any set of topics to be placed on a broker. However, building such a prediction model first requires a critical understanding of the impact of pub/sub features and topic co-location on per-topic latencies. Therefore, we first conduct a set of sensitivity analysis experiments to

[1]Otherwise, the topic must be split into two or more topics, e.g., by splitting publishing rate [22] for any solution to be feasible. The design of topic splitting policies is out of the scope of this paper.



(a) Impact of subscription size

(b) Impact of topic co-location (one-to-many)

(c) Impact of publishing rate

(d) Impact of topic co-location (many-to-one)

Fig. 1: Sensitivity analysis for latency modeling

study the impact of several pub/sub features, such as number of subscribers or subscription size, publishing rate, per-sample processing interval and background load on a topic's performance. This helps us to identify the dominant pub/sub features that should be used to build the latency prediction model.

Accordingly, we first describe our pub/sub system and the experimental testbed used to conduct the sensitivity analysis experiments in Section IV-A, following which Section IV-B presents the sensitivity analysis results. Section IV-D describes our latency prediction model, followed by discussions about its limitations in Section IV-E.

A. Experimental Setup

We implemented our pub/sub system using the Java language binding of the ZMQ (http://zeromq.org/) sockets library. Our system architecture is similar to Kafka, where topics are hosted on a flat layer of pub/sub brokers managed by Zookeeper (https://zookeeper.apache.org/), which is a centralized service for distributed coordination and state maintenance. Publishers and subscribers connect to the broker that hosts their topics of interest to send and receive data, respectively. We have used the matrix-product CPU stressor provided by the stress-ng (http://kernel.ubuntu.com/~cking/stress-ng/) tool to emulate variable intervals of per-sample processing performed at the broker in accordance with the publish-process-subscribe paradigm. All experiments are performed for a broker node hardware with four 2.5GHz Intel Xeon E5420 cores, 4GB RAM and 1Gb/sec network capacity. Separate machines were used for hosting the publisher and subscriber endpoints. Network Time Protocol (NTP)[2] was used for time synchronization of all machines. Publishers tag their messages with a time-stamp and subscribers upon reception of the message use this time-stamp to compute the end-to-end latency of data delivery.

B. Sensitivity Analysis

In our motivational use-case described in Section III-A, IoT workload is either of type one-to-many or of type many-to-one. In the one-to-many type, a single/few publishers send processed data to a large number of subscribers for actuation. For example, in our motivational use case, taxis receive information about the most profitable region of operation on the *update* topic. In the many-to-one type, a large number of publishers send their data to a few subscribers for processing. For example, all taxis in a region send their gps coordinates on the *gps* topic. The publishing rate in the one-to-many case is expected to be low, while the cumulative publishing rate for the many-to-one case is expected to be much higher.

We first study the impact of number of subscribers on a topic's performance for one-to-many type of data dissemination (recall that a topic's performance is characterized by the end-to-end 90th percentile latency experienced by its subscribers). Figure 1a shows the impact on latency when we increase the number of subscribers connected to a topic hosted at a broker in isolation (i.e., there are no other topics

[2]http://www.ntp.org/

hosted along with this topic at the broker) for different values of per-sample processing interval. As this is a one-to-many type of workload, we connect a single publisher, which sends messages with a payload size of 4KB at a low rate of 1 message/second. We observe that although latency increases with increasing number of subscribers, the impact on latency is very small, especially if sub-second delivery bounds are considered, such as in our taxi use case. From our analysis of the New York taxi data-set over a five-day period, the maximum number of taxis in a region was found to be 240. Figure 1a shows that 240 subscribers/taxis can easily be sustained without incurring a significant performance penalty.

However, a topic will seldom be hosted in isolation at the broker given the resource-constrained nature of edge-based systems. Therefore, we also study how co-located topics can impact the performance of a topic of type one-to-many. We refer to the topic under consideration as the *foreground* topic, while the other co-located topics are referred to as the *background* topics. Figure 1b shows how the latency of the foreground topic with 200 connected subscribers and per-sample processing of 50ms is affected as we increase the number of background topics. Here, each background topic is of type many-to-one with 10 connected publishers- each publishing at the rate of one message/second, one connected subscriber and 50ms per-sample processing interval. The number of background topics is increased until CPU utilization at the broker saturates. We see that the latency of the foreground topic increases significantly with increasing load at the broker, but it is still well below the sub-second latency bound despite broker CPU saturation.

We perform similar sensitivity analysis experiments for the many-to-one scenario. Figure 1c shows how an isolated topic's latency is impacted as publishing rate or number of connected publishers increases for different per-sample processing intervals. Here, each topic has one connected subscriber and each publisher publishes at the rate of one message/second. We observe that a topic's latency increases linearly up to a threshold rate after which the increase in latency becomes exponentially large. Beyond the threshold publishing rate, the processing capacity of the broker is exceeded by the incoming rate of messages, which results in large queuing delay at the broker and therefore, an exponential increase in the observed latency [42]. The threshold rate for a given per-sample processing interval decreases due to the impact of background load imposed by other co-located topics as shown in Figure 1d. This shows that the threshold rate for a topic with per-sample processing interval of 20ms reduces from 38 messages/second in the isolated case to 15 messages/sec when co-located with 7 other background topics and to 8 messages/second when co-located with 11 other background topics. In these experiments, the background topics were of type many-to-one, with randomly chosen publishing rates and per-sample processing intervals.

C. Key Insights from Sensitivity Analysis

The sensitivity analysis results show that a topic's latency in a publish-process-subscribe system increases with increasing subscription size, publishing rate, per-sample processing interval and background load. Depending on the broker hardware capacity, the measured values, such as, latency, threshold rate of publication, etc. may be different. However, observed behaviors will remain the same.

The sensitivity analysis experiments show that number of subscribers, publishing rate, per-sample processing interval and background load all impact a topic's latency. For sub-second latency requirements, the results show that even for 200 connected subscribers, the latency for a topic is not significantly impacted despite the broker being saturated. For applications where the number of connected subscribers per topic is much larger than 200, a topic can be replicated and the number of connected subscribers can be distributed [22] to ensure that the latency QoS is met. Topic partitioning and replication is beyond the scope of this paper and we assume that a topic can be safely placed at a broker in isolation. For sub-second latency bounds, as the number of subscribers does not significantly impact latency, we did not include it in the latency prediction model. It is important to note, however, that the number of subscribers may need to modeled for systems with stricter latency requirements. Figure 1b shows a 147% increase in latency from 72ms to 180ms as the background load increases for the foreground one-to-many topic with 200 connected subscribers.

D. Latency Prediction Model

The sensitivity analysis experiments show that a topic's per-sample processing interval and publishing rate prominently impact its latency. Hence, we have considered these two pub/sub features and features derived from them for learning our latency prediction model. More concretely, we used six input features for learning the model, of which the first three characterize the foreground topic and the remaining three characterize the background load.

These input features are described below, where t_f denotes the foreground topic and T_B denotes the set of background topics at a broker:

- p_f, i.e., per-sample processing interval of t_f;
- r_f, i.e., publishing rate of t_f;
- d_f, i.e., foreground load which is the product of per-sample processing interval p_f and publishing rate r_f;
- $\sum_{t_b \in T_B} p_b$, i.e., the sum of per-sample processing intervals of all background topics;
- $\sum_{t_b \in T_B} r_b$, i.e., the sum of publishing rates of all background topics;
- $\sum_{t_b \in T_B} d_b$, i.e., the sum of load (product of per-sample processing interval and publishing rate) of all background topics.

For a topic in isolation, the background load is zero. Therefore, to learn the latency model for a topic in isolation, we have only considered the per-sample processing interval p

and publishing rate r of the topic as input features. We found that polynomial regression of degree 4 accurately models the latency curve for a topic in isolation. We describe the isolated topic model accuracy results in more detail in Section VI-B.

However, for two or more co-located topics, i.e., $k \geq 2$, we found that simple regression techniques could no longer capture the complex, non-linear impact of the background topics' loads on a foreground topic's latency. Since neural networks generally perform well in capturing non-linear functions of a problem [14], we use it to learn latency models for $k \geq 2$. Neural networks comprise multiple layers, namely, an input layer, one or more intermediate layers called hidden layers and an output layer. Each layer comprises nodes called neurons which are linked to neurons in another layer by weighted connections. The input layer simply feeds the input features of the training data to the network via these weighted connections. Neurons of the hidden and output layer sum the incoming weighted input signals, apply a bias term and an activation function to produce the output for the next layer (in case of a hidden layer) or the output of the network (in case of the output layer). Hidden layers and/or non-linear activation functions are used for modeling the non-linearity of the problem.

The architecture of a neural network, specifically the number of hidden layers, number of neurons in each hidden layer and the regularization factor, greatly impact the performance of the model. If the chosen architecture is too complex, it may result in over-fitting the data and may not generalize to perform well outside of the training data. In this case, training error is very low, but the error on the validation data is high and the model is said to suffer from high variance [7]. On the other hand, if the chosen architecture is too simple, it fails to learn from the data (under-fitting) and performs badly on both the training and validation data. In this case, the model is said to suffer from high bias [7]. Learning curves [35], which plot the training and validation errors as functions of the training data size can be used to select the right architecture for reducing both the bias and the variance of the model. Specifically, the network architecture for which both training and validation errors converge to a low value is typically chosen.

We learn a separate k-topic co-location model using neural networks for each $k \geq 2$ as opposed to a unified model for reasons of higher accuracy. We plot learning curves for several different neural network architectures for each k and select the one which minimizes both bias and variance. While simpler network architectures perform well for lower values of k, more complex architectures are needed for higher values of k as the search space increases. Section VI-B shows the learning curves and accuracy of the learned models.

E. Limitations of the Model

It is important to note the limitations of the k-topic co-location models. The learned latency models are specific to a broker hardware type. Therefore, separate latency models need to be learned for each new hardware type. Model learning overhead for different hardware architectures can be reduced by incorporating hardware-specific input features in the learned models so that a single model can be used across different architectures. Transfer learning [37] can also be used for learning the latency models for different broker architectures on the basis of existing models for a specific hardware architecture. Finally, our model also assumes that the per-sample processing performed at each topic is CPU-bound. Incorporating different types of processing loads, such as memory, disk or network bound in the latency model is our planned future work.

The learned latency model is used by our topic placement heuristics for k-TCP (see Section V) so that the latency QoS of the topics is not violated on a broker. However, subject to the inaccuracy of the latency prediction model, the produced placement for some topics in the system may be incorrect and may result in QoS violation for those topics. Hence, our approach does not provide hard guarantees on meeting the specified latency QoS. To address this issue, our approach can be augmented with a feedback-based mechanism where subscribers experiencing QoS violations can inform the system, which can then place this topic on another broker and also use this information to update the learned latency model. Employing a latency model as opposed to relying solely on a subscriber's feedback has the following benefits: 1) QoS violations can be prevented proactively for most of the cases where the latency model makes accurate predictions; and 2) subscriber feedback-based mechanism can incur a large overhead as the system scales.

V. NP-Completeness of k-TCP and Heuristics-based Solutions

In this section, we analyze the computational complexity of k-TCP and show that it is NP-hard for $k \geq 3$, which represents a fairly small degree of co-location. We then propose some heuristics to solve k-TCP sub-optimally.

A. Feasibility Function

Before analyzing the complexity and proposing a solution for k-TCP, we first rely on the latency prediction model to define a *feasibility function* $\mathcal{F}$, which indicates whether a given set of at most k topics can be feasibly co-located on a broker. Specifically, for any $T^{'} \subseteq T$ and $|T^{'}| \leq k$, we can rely on the k' co-location model for $k' = |T^{'}|$ to predict the latencies for all topics in $T^{'}$, while using the individual parameters for each topic, i.e., the per-sample processing interval p_i and publishing rate r_i, as the input features for the model as described in Section IV-D. We define:

$$\mathcal{F}(T^{'}) = \begin{cases} 1 & \text{if } \ell_i(T^{'}) \leq \tau \text{ for all } t_i \in T^{'} \\ 0 & \text{otherwise} \end{cases}$$

Hence, according to Property 1, we have:

$$\mathcal{F}(T^{'}) = 1 \text{ for any } |T^{'}| = 1 \tag{5}$$

$$\mathcal{F}(T^{'}) = 1 \text{ implies } \mathcal{F}(T^{''}) = 1 \text{ for any } T^{''} \subseteq T^{'} \tag{6}$$

Note that, for a constant degree of co-location k, the set of all possible inputs to the feasibility function $\mathcal{F}$ can be encoded

by at most $\sum_{k'=1}^{k} \binom{n}{k'} = O(n^k)$ bits, i.e., polynomial in the number of topics.

B. Complexity Analysis

We now show the computational complexity of k-TCP.

Theorem 1. *For $k \leq 2$, k-TCP can be solved in polynomial time.*

Proof. The claim is obvious for $k = 1$ (i.e., 1-TCP). In this case, each topic must be assigned to a broker alone, and the number of required brokers is therefore $|B| = |T|$.

For $k = 2$ (i.e., 2-TCP), construct a graph $G = (V, E)$, where $|V| = |T|$ and each vertex $v_i \in V$ represents a topic $t_i \in T$. An edge e_{ij} exists between two vertices v_i and v_j if the corresponding two topics t_i and t_j can be feasibly co-located, i.e., $\mathcal{F}(\{t_i, t_j\}) = 1$. Finding a maximum matching M of G, which can be computed in polynomial time [38], will lead to an optimal solution, where each pair of matched vertices (topics) are co-located on a broker and the unmatched ones are each assigned to a broker alone. The optimal number of required brokers is in this case $|B| = |T| - |M|$. □

Theorem 2. *For $k \geq 3$, k-TCP is NP-hard.*

Proof. We prove the NP-completeness for the decision version of k-TCP, which for a given instance asks whether the collection of topics can be co-located on m or fewer brokers. The problem is clearly in NP: given a co-location scheme, we can verify in polynomial time that it takes at most m brokers and that the set of co-located topics on any broker has a cardinality at most k and form a feasible set (via the feasibility functions).

To show that the problem is NP-complete, we use a reduction from k-Dimensional Matching (k-DM), which is a generalization of the well-known 3-Dimensional Matching (3-DM) problem. For k-DM, we are given k disjoint sets $X_1, X_2, \ldots, X_k$, where all X_j's have the same number m of elements. Let M be a subset of $X_1 \times X_2 \times \cdots \times X_k$, that is, M consists of k-dimensional vectors $(x_1, x_2, \ldots, x_k)$ such that $x_j \in X_j$ for all $1 \leq j \leq k$. The question is whether M contains a perfect matching $M' \subseteq M$, that is, $|M'| = m$, and for any distinct vectors $(x_1', x_2', \ldots, x_k') \in M'$ and $(x_1'', x_2'', \ldots, x_k'') \in M'$, we have $x_j' \neq x_j''$ for all $1 \leq j \leq k$. It is known k-DM is NP-complete for $k \geq 3$ [21].

Given an instance I_1 of k-DM, we construct an instance I_2 of k-TCP by creating km topics, each corresponding to an element in I_1. For each vector $(x_1, x_2, \ldots, x_k) \in M$ in I_1, we set the corresponding set of topics to be feasible in I_2, i.e., $\mathcal{F}(\{x_1, x_2, \ldots, x_k\}) = 1$, and derive the other feasible sets using Equations (5) and (6) while leaving all the remaining sets to be infeasible. Finally, the bound on the number of brokers in I_2 is set to be m. Clearly, if I_1 admits a perfect matching of size m, then we can co-locate the corresponding sets of topics in I_2 using m brokers. On the other hand, if all topics in I_2 can be co-located using m brokers, since there are km topics in total and each broker can accommodate at most k topics, then each broker must contain exactly k distinct topics, which based on the reduction must come from the elements in one of the k-dimensional vectors of I_1. Thus, using the sets of co-located topics in I_2, we can get a perfect matching M' for I_1. □

C. Heuristics

Given the NP-hardness result for $k \geq 3$, we propose heuristic solutions to solve k-TCP sub-optimally. Recall that the goal is to find a co-location scheme for a collection T of topics on a minimum set B of brokers. Equivalently, the topics could be considered to form a partition of B disjoint subsets $\{T(b_1), T(b_2), \ldots, T(b_{|B|})\}$ such that each broker b_j hosts a feasible subset $T(b_j) \subseteq T$ of topics, i.e., $\bigcup_{b_j \in B} T(b_j) = T$ and $T(b_j) \bigcap T(b_{j'}) = \emptyset$ for any $b_j \neq b_{j'}$.

In the following, we first describe two heuristics that are inspired by the greedy algorithms in bin packing and set cover problems, respectively, and apply them to the k-TCP context. We then present a hybrid heuristic that combines the two algorithms.

a) First Fit Decreasing*:* The first heuristic is inspired by a greedy algorithm in the bin packing problem, which we call First Fit Decreasing, or FFD$_k$ for a given degree of co-location k, and its pseudocode is presented in Algorithm 1. First, the algorithm sorts all topics in decreasing order of latency when they are assigned to a broker in isolation (line 2). Then, it considers each topic in sequence and finds the first broker that can feasibly host it together with the existing topics that have already been assigned to the broker (lines 6-15). If no such broker can be found, it starts a new broker and assigns the topic there (lines 16-19), which according to Property 1(a) is always feasible. The complexity of the algorithm is $O(n \log n + n|B|)$, where $|B|$ is the total number of brokers in the solution. Since $|B| \leq n$, the algorithms runs in $O(n^2)$ time in the worst case.

b) Largest Feasible Set*:* The second heuristic is inspired by the greedy algorithm in the set cover problem and the set packing problem. We call it Largest Feasible Set, or LFS$_k$ for a given degree of co-location k, and its pseudocode is presented in Algorithm 2. Specifically, the algorithm works in iterations. At each iteration, it finds any largest feasible set of k topics and co-locates them on a new broker (lines 4-8). If no such set can be found anymore, the maximum degree of co-location k is then decremented by 1 (line 9), and the process continues until k is reduced down to 2, in which case we can run the maximum matching algorithm (lines 10-12) as described in the proof of Theorem 1 that guarantees to co-locate the remaining topics in an optimal fashion. The complexity of the algorithm is $O(n^k)$ dominated by enumerating all possible subsets of k topics in the worst case for the feasibility test. Note that although the complexity is polynomial in the number n of topics, the running time can be prohibitive for a high degree of co-location (e.g., $k > 4$) on even moderate n. We resolve this problem below with a hybrid heuristic.

c) A Hybrid Solution*:* We now present a heuristic that combines the benefits of top-down search of LFS and low complexity of FFD. In particular, the algorithm takes a parameter $k' \leq k$ as input, and Algorithm 3 shows its pseudocode. We call the algorithm LFS$_{k'}$+FFD$_k$. Similarly to LFS$_k$, this

Algorithm 1: FirstFitDecreasing (FFD$_k$)

Input: Collection $T = \{t_1, t_2, \ldots, t_n\}$ of n topics, latency ℓ_i for each topic $t_i \in T$ when assigned to a broker in isolation, degree of co-location k, and feasibility function $\mathcal{F}$
Output: A partition of topics $\{T(b_1), T(b_2), \ldots, T(b_{|B|})\}$ for a set B of brokers with each broker $b_j \in B$ hosting a subset $T(b_j) \subseteq T$ of topics

```
1  begin
2      Sort the topics in decreasing order of latency when assigned to a
         broker in isolation, i.e., ℓ_1 ≥ ℓ_2 ≥ ··· ≥ ℓ_n;
3      Initialize |B| ← 0;
4      for topic t_i (i = 1 ... n) do
5          mapped ← false;
6          for broker b_j (j = 1 ... |B|) do
7              if |T(b_j)| = k then
8                  continue;
9              end
10             if F(T(b_j) ⋃ {t_i}) = 1 then
11                 T(b_j) ← T(b_j) ⋃ {t_i};
12                 mapped ← true;
13                 break;
14             end
15         end
16         if mapped = false then
17             |B| ← |B| + 1;
18             Start a new broker b_|B| with T(b_|B|) = {t_i};
19         end
20     end
21 end
```

Algorithm 2: LargestFeasibleSet (LFS$_k$)

Input: Collection $T = \{t_1, t_2, \ldots, t_n\}$ of n topics, degree of co-location k, and feasibility function $\mathcal{F}$
Output: A partition of topics $\{T(b_1), T(b_2), \ldots, T(b_{|B|})\}$ for a set B of brokers with each broker $b_j \in B$ hosting a subset $T(b_j) \subseteq T$ of topics

```
1  begin
2      Initialized |B| ← 0;
3      while T ≠ ∅ do
4          while ∃T' ⊆ T s.t. F(T') = 1 and |T'| = k do
5              |B| ← |B| + 1;
6              Start a new broker b_|B| with T(b_|B|) = T';
7              T ← T\T';
8          end
9          k ← k − 1;
10         if k = 2 then
11             MaximumMatching(T);
12         end
13     end
14 end
```

hybrid solution also works in iterations, but an iteration now consists of two steps. In the first step, any feasible set of k' topics (if any) are found and co-located on a new broker (lines 4-7). This is followed by the second step, which uses the first fit heuristic to maximize the degree of co-location up to k on this broker (lines 8-16) based on a sorted sequence of the remaining topics (line 2). This two-step iteration continues until no feasible set of k' topics can be found. In this case, the algorithm resorts to LFS with parameter $k' - 1$ for assigning the remaining topics (line 18). The overall complexity of the algorithm is $O(n^{k'} + n^2/k')$, with the two parts coming from running LSF initially (using parameter k') and FFD (on at most n/k' brokers), respectively. Note that when the parameter

Algorithm 3: LFS$_{k'}$+FFD$_k$

Input: Collection $T = \{t_1, t_2, \ldots, t_n\}$ of n topics, latency ℓ_i for each topic $t_i \in T$ when assigned to a broker in isolation, degree of co-location k, parameter $k' \leq k$, and feasibility function $\mathcal{F}$
Output: A partition of topics $\{T(b_1), T(b_2), \ldots, T(b_{|B|})\}$ for a set B of brokers with each broker $b_j \in B$ hosting a subset $T(b_j) \subseteq T$ of topics

```
1  begin
2      Sort the remaining topics in decreasing order of latency when
         assigned to a broker in isolation;
3      Initialized |B| ← 0;
4      while ∃T' ⊆ T s.t. F(T') = 1 and |T'| = k' do
5          |B| ← |B| + 1;
6          Start a new broker b_|B| with T(b_|B|) = T';
7          T ← T\T';
8          for topic t_i (i = 1 ... |T|) do
9              if |T(b_|B|)| = k then
10                 break;
11             end
12             if F(T(b_|B|) ⋃ {t_i}) = 1 then
13                 T(b_|B|) ← T(b_|B|) ⋃ {t_i};
14                 T ← T\{t_i};
15             end
16         end
17     end
18     LargestFeasibleSet(T, k' − 1);
19 end
```

satisfies $k' = k$ the algorithm becomes exactly LFS$_k$, and when $k' = 1$ it becomes FFD$_k$. Thus, for a suitable choice of k', the algorithm combines the two previous heuristics while offering a lower complexity solution to the problem.

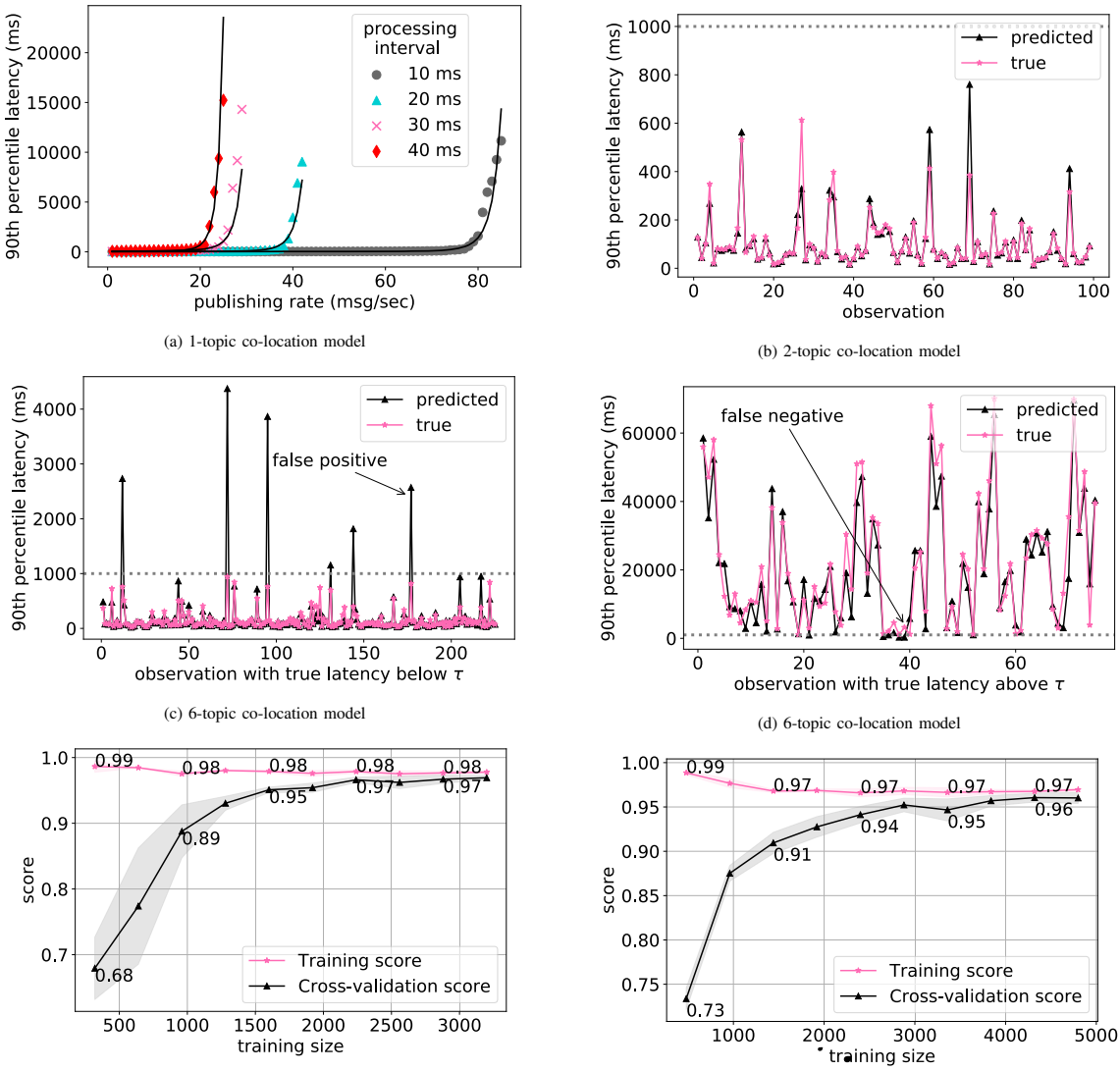
VI. Experiments

In this section, we present experimental results to validate our proposed solution for providing latency QoS of data delivery in publish-process-subscribe systems. We first describe the testbed used for conducting the experiments, and then present the accuracy results of the *k*-topic co-location model and the performance results for the proposed *k*-TCP heuristics.

A. Experimental Testbed and Setup

Our testbed comprises 25 heterogeneous machines running Ubuntu 14.04, of which 13 are homogeneous machines with four 2.5GHz Intel Xeon E5420 cores, 4GB RAM and 1Gb/s network adapter, which were used for running the brokers. The *k*-topic co-location models are learned for this hardware type. The remaining machines were used to host the publisher/subscriber endpoints and the Zookeeper coordination service. We benchmarked the machines used to run the endpoints to find the maximum number of endpoints that can be run on them reliably. This was done to minimize the effect of resource contention on the experimental results. All machines were time synchronized using NTP.

Drawing from our motivational use case (Section III-A) and the RIoTBench [43] results in which the ETL and prediction stream processing pipelines for the New York taxi data were benchmarked to take between 10ms and 40ms, the per-sample processing interval for any topic in our experiments was set to be either 10ms, 20ms, 30ms or 40ms. Publisher endpoints



(a) 1-topic co-location model

(b) 2-topic co-location model

(c) 6-topic co-location model

(d) 6-topic co-location model

(e) 4-topic co-location learning curve

(f) 6-topic co-location learning curve

Fig. 2: Performance of latency prediction model

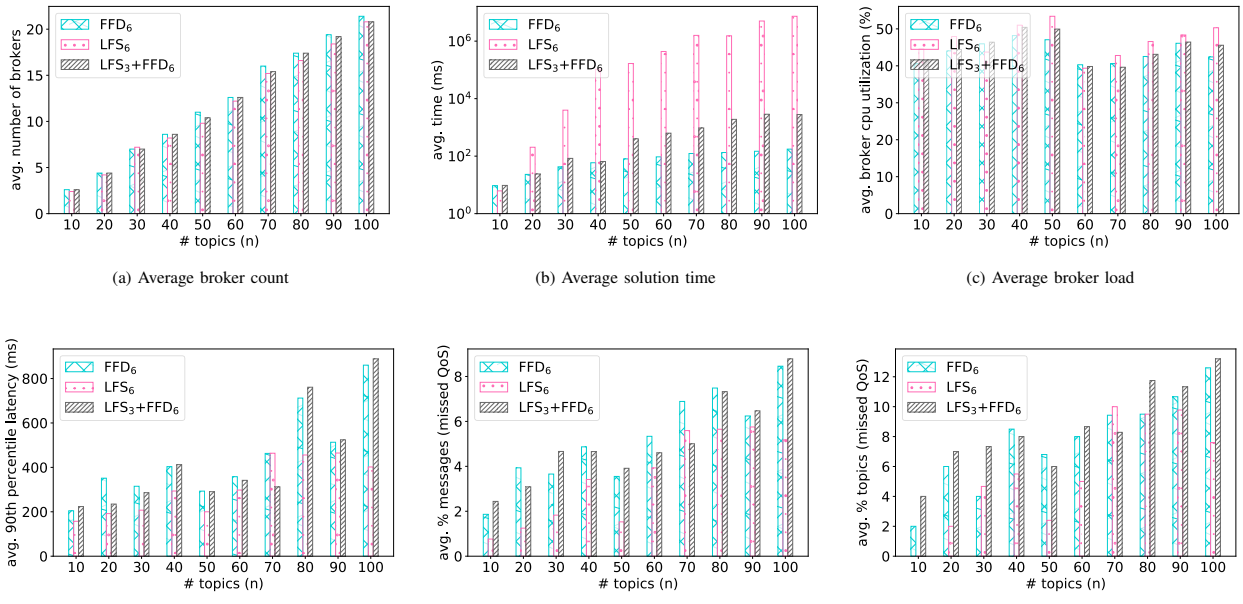
send 4KB messages at the rate of 1 message/second for two minutes (i.e., a total of 120 messages per publisher) to ensure that any experiment runs for a reasonable length of time. If a topic t in an experiment is configured with per-sample processing interval p and publishing rate r, then the broker is configured to execute `stress-ng matrix-product` so that it takes p per-sample processing time. Additionally, one subscriber and r publisher endpoints for topic t are created. All subscriber and publisher endpoints connect to the system before starting the experiment to ensure the fidelity of experimental results. In computing the 90th percentile latency of a topic, the latency values for some initial messages on the topic are not considered since they are observed to be very high due to initialization and connection setup.

TABLE I: Accuracy of k-topic co-location model

k	#datapoints (training)	accuracy (training)	accuracy (test)	#datapoints (validation)	accuracy (validation)
2	2000	.987	.985	100	.972
3	3000	.985	.978	150	.976
4	4000	.983	.979	200	.984
5	5000	.981	.978	250	.951
6	6000	.981	.956	300	.968

B. k-Topic Co-location Model Learning

In order to learn the k-topic co-location model for $k \geq 2$, we first learn the latency model for a topic in isolation, i.e., when $k = 1$. In particular, the 1-topic co-location model takes the per-sample processing interval p and publishing rate r of



(a) Average broker count

(b) Average solution time

(c) Average broker load

(d) Average 90th percentile latency

(e) Average %messages (missed QoS)

(f) Average %topics (missed QoS)

Fig. 3: Performance of k-TCP heuristics for varying n

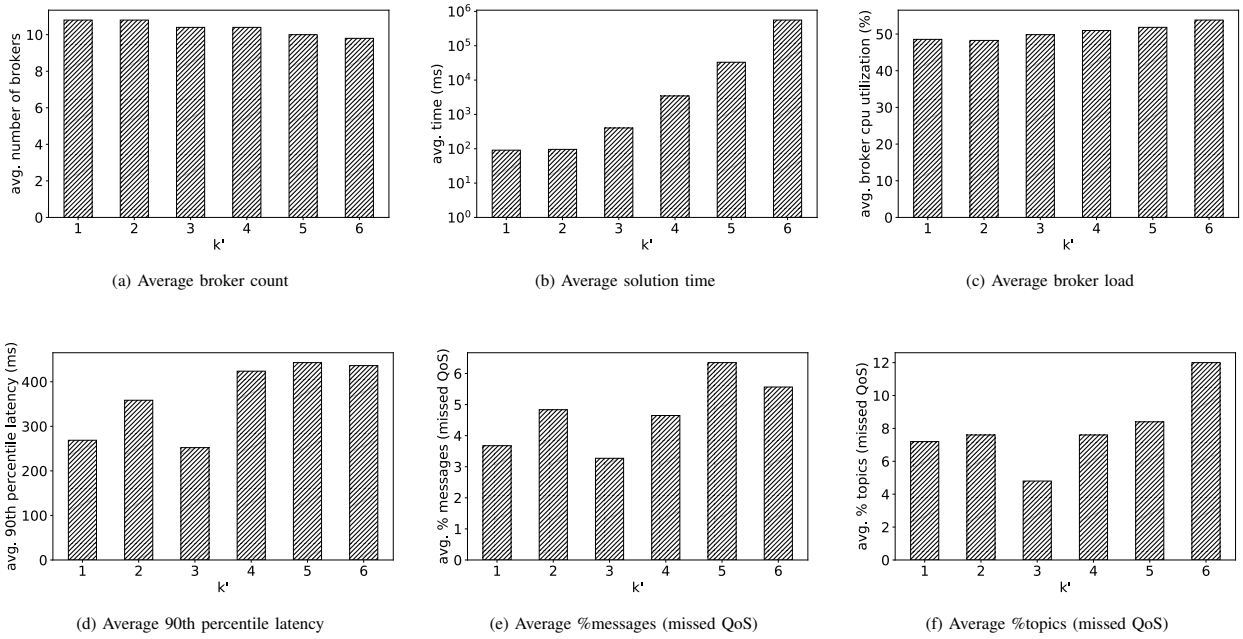
a topic t as inputs and predicts its 90th percentile latency ℓ. We can use this model to estimate the maximum sustainable publishing rate $r_{\max}$ for the topic with per-sample processing time p beyond which the 90th percentile latency for the topic will violate its desired QoS τ. This maximum rate $r_{\max}$ is used to ensure Property 1(a), i.e., a topic can always be placed on a broker in isolation, otherwise topic replication and partitioning of publishers over topic replicas [22] will be needed.

We found that polynomial regression of degree 4 provides the best fit for the 1-topic co-location model with a training and test accuracy of .975 and .97, respectively. We used a dataset with 180 datapoints; 60% of which were used for training and the remaining 40% were used for testing. Figure 2a shows the fit of the polynomial curve in degree 4 over experimentally observed 90th percentile latency values. Here, the x-axis shows the publishing rate r in messages/second and the y-axis shows the 90th percentile latency in milliseconds for p values of 10ms, 20ms, 30ms and 40ms. Using the model, we found the $r_{\max}$ for sub-second 90th percentile latencies to be 78 messages/second, 37 messages/second, 24 messages/second and 20 messages/second for p values of 10ms, 20ms, 30ms and 40ms, respectively.

We then used $r_{\max}$ found under the 1-topic co-location model to create the training dataset for k-topic co-location models with $k \geq 2$. To create the training dataset, for each topic, p was uniformly randomly chosen from the set $\{10\text{ms}, 20\text{ms}, 30\text{ms}, 40\text{ms}\}$ and r was uniformly randomly chosen from the range $[1, r_{\max}]$. For each k-topic co-location model, we trained over 1,000 different randomly generated test configurations, and each configuration contains k datapoints, one for each of the k topics. This gives $1000k$ datapoints for each k-topic co-location model. A test runs for $\sim$3 mins and it took $\sim$11 days to collect the training data to learn these offline latency models. In all of these experiments, the network utilization was kept well below the 1Gb/sec network capacity of the broker to make sure that network saturation does not impact the gathered results.

We tested different neural network architectures for each k-topic co-location model, and found that a neural network with two hidden layers composed of 40 neurons each performed well for $k \leq 5$. Figure 2e shows the learning curve for $k = 4$. The learning curve shows that the chosen neural network architecture has low bias and variance since both training and validation errors converge to a low value of $\sim$3%. A more complex neural network architecture was needed for $k = 6$ as the parameter space increases. In this case, a neural network with two hidden layers composed of 100 neurons each performed well. Figure 2f shows the learning curve for the 6-topic co-location model. Again, we see that the chosen neural network architecture has both low bias and low variance.

As described in Section IV, the input features for the model are p_f, r_f, d_f, $\sum_{t_b \in T_B} p_b$, $\sum_{t_b \in T_B} r_b$ and $\sum_{t \in T_B} d_b$. Table I shows the accuracy of the learned models for k up to 6. We used the logarithm of the 90th percentile latency as the output for the model as it performed better than using the 90th percentile latency value itself. Rectified Linear Units (ReLu) was used as the activation function, the limited memory Broyden-Fletcher-Goldfarb-Shanno (`lbfgs`) solver was used and the L2 regularization factor was set to 0.1. We used 95% of the datapoints for training and the remaining 5% for testing. A separate validation dataset was created by running 50 different test configurations for each k. The performance of the learned



(a) Average broker count

(b) Average solution time

(c) Average broker load

(d) Average 90th percentile latency

(e) Average %messages (missed QoS)

(f) Average %topics (missed QoS)

Fig. 4: Performance of LFS$_{k'}$+FFD$_k$ for varying k'

models on the validation dataset is also shown in Table I. We see that the learned models have an accuracy of ∼97%.

Figure 2b shows the performance of the 2-topic co-location model on the validation dataset. We see that the predicted latency tracks the experimentally observed latency values closely. Similarly, Figure 2c and Figure 2d show the performance of the 6-topic co-location model on test data points for which the experimentally observed latency values are below and above τ, respectively. There are cases where the model makes inaccurate latency predictions, resulting in both false-positives and false-negatives. Figure 2c shows a false-positive occurrence where the predicted value is greater than τ and the experimentally observed 90th percentile latency is below τ. Figure 2d shows a false-negative occurrence where the predicted 90th percentile latency is below τ and the experimentally observed 90th percentile latency is above τ. False negatives result in QoS violations and false positives result in inefficient resource utilization. Further improvement in the accuracy of the *k*-topic co-location model by using advanced machine learning methods is part of our planned future work.

C. Performance of k-TCP Heuristics

We now study how each of the three k-TCP heuristics, namely FFD$_k$, LFS$_k$, and LFS$_{k'}$+FFD$_k$, perform for $k = 6$ as the number of n topics to be placed increases. We set the parameter k' of the hybrid heuristic to be $k' = 3$. In Figure 3, we present results averaged over 5 random placement requests for each value of n. The performance of these heuristics is compared along the following six dimensions: 1) number of brokers needed for hosting n topics; 2) time to find a placement solution for n topics; 3) average CPU utilization of all brokers used for hosting n topics; 4) average 90th percentile latency of all n topics; 5) percentage of all messages across n topics with latency greater than τ (1 second); and 6) percentage of n topics whose 90th percentile latency is greater than τ.

In Figure 3a, we observe that the three heuristics perform similarly to each other in terms of the number of brokers used for placing the topics. LFS$_k$ is able to find a placement which uses less number of brokers than both FFD$_k$ and LFS$_{k'}$+FFD$_k$ for most of the cases. However, as seen in Figure 3b, LFS$_k$ takes a much longer time to find a placement than FFD$_k$. As expected, LFS$_{k'}$+FFD$_k$, being a hybrid of the other two, takes less time than LFS$_k$ but more time than FFD$_k$. Average CPU load of the brokers in the system for the placement produced by FFD$_k$, LFS$_k$ and LFS$_{k'}$+FFD$_k$ as seen in Figure 3c, does not show a wide variation.

Figure 3d shows the average 90th percentile latency across all n topics in the system for the placements produced by the three heuristics. FFD$_k$ and LFS$_{k'}$+FFD$_k$ have comparable performance in most cases, while LFS$_k$ yields a lower average 90th percentile latency for all values of n. Figure 3e shows that up to 9% of all messages in the system are not able to meet their latency QoS. The percentage of messages that miss their QoS is comparable for both FFD$_k$ and LFS$_{k'}$+FFD$_k$ in most cases, while LFS$_k$ performs better with a lower percentage of messages with QoS violations. Similarly in Figure 3f, we see that the percentage of topics that miss their QoS is comparable for FFD$_k$ and LFS$_{k'}$+FFD$_k$ in most cases, while LFS$_k$ yields a lower percentage of topics with missed QoS in almost all cases except for $n = 70$. It shows that up to 13% of the

topics in the system miss their QoS due to incorrect broker assignment, and we are able to meet the QoS requirements for 87% of the topics in the system. As discussed in Section IV, our solutions can be used along with a subscriber feedback mechanism to place the topics experiencing QoS violation on another broker.

These results show that while LFS_k heuristic performs better than FFD_k and $\text{LFS}_{k'}$+FFD_k, it has a prohibitively large running time. On the other hand, FFD_k takes much less time to compute the placement and performs comparably well with $\text{LFS}_{k'}$+FFD_k. Hence, by tolerating some degradation in performance, simpler heuristics such as FFD_k can be employed in favor of computationally more expensive heuristics like LFS_k.

D. Performance of $LFS_{k'}$+FFD_k

Finally, we study how the hybrid heuristic $\text{LFS}_{k'}$+FFD_k performs with varying value of k'. As discussed earlier in Section V, $\text{LFS}_{k'}$+FFD_k behaves as FFD_k for $k' = 1$ and as LFS_k for $k' = k$. Figure 4 shows the results when k' varies from 1 to 6. For each value of k', we present results averaged over the same 5 random placement requests for $n = 50$ topics.

As expected, for higher values of k', $\text{LFS}_{k'}$+FFD_k finds a placement that uses fewer brokers, as seen in Figure 4a. However, the time to find the solution also increases with k' as seen in Figure 4b. Average CPU utilization of the brokers does not show much variation for different values of k', as seen in Figure 4c. Average 90th percentile latency increases for higher values of k' as seen in Figure 4d, since the placement of topics produced is more compact. Once again, the QoS requirement is not always met in the placement solution produced by the hybrid heuristic. Up to 6% of the messages and up to 12% of the topics experience QoS violations as seen in Figure 4e and Figure 4f.

VII. Conclusion and Discussions

Many emerging IoT applications are latency critical in nature and require both real-time data dissemination and information processing. The Publish/Subscribe (pub/sub) pattern for many-to-many communications is often used to meet the scalable data dissemination needs of IoT applications. With the emergence of edge computing that promotes processing near the source of data, the pub/sub system has been extended to support processing at the edge-based pub/sub brokers, making it the *publish-process-subscribe* pattern. It is in this context that end-to-end quality of service (QoS) for data dissemination and processing must be satisfied to realize the next generation of edge-based, performance-sensitive IoT applications.

This paper presents a solution to provide the desired latency QoS for data dissemination and processing in topic-based publish-process-subscribe systems. Our proposed solution learns a latency prediction model for a set of co-located topics on an edge broker and uses this model to balance the processing and data-dissemination load to provide the desired QoS. Specifically, the paper made the following contributions: (a) a sensitivity analysis on the impact of different pub/sub features including the number of subscribers, number of publishers, publishing rate, per-sample processing interval and background load, on a topic's 90th percentile latency; (b) a latency prediction model for a topic's 90th percentile latency, which was then used for the latency-aware placement of topics on brokers; and (c) an optimization problem formulation for k-topic co-location to minimize the number of brokers used while providing QoS guarantees.

The following lessons were learned from and insights into different dimensions of future work were informed by this research:

- The accuracy of the *k*-topic co-location model has a significant impact on QoS satisfaction and resource efficiency. More advanced machine learning methods for learning the *k*-topic co-location model could be investigated. For higher values of *k*, training over a larger search space is needed for good accuracy, but this will require significant additional resources for model learning. Online methods for updating the prediction model can also be explored, e.g., via reinforcement learning [29]. Transfer learning [37] of the k-topic co-location models for different hardware architectures on the basis of some learned models is another direction that can be explored.
- Currently, our load balancing decisions are made statically including the topic placement decisions. Future work will therefore involve dynamic load balancing decisions including elastic auto-scaling of the number of brokers used. Proactive provisioning of resources can also be performed on the basis of workload forecasting. Finally, network link state can also be incorporated.

The source code and experimental apparatus used in the research is made available in open source at https://github.com/doc-vu/edgent.

Acknowledgments

This work is supported in part by NSF US Ignite CNS 1531079 and AFOSR DDDAS FA9550-18-1-0126. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF or AFOSR.

References

[1] "Averages can be misleading: try a percentile," https://www.elastic.co/blog/averages-can-dangerous-use-percentile.

[2] S. Abdelwahab and B. Hamdaoui, "Fogmq: A message broker system for enabling distributed, internet-scale iot applications over heterogeneous cloud platforms," *arXiv preprint arXiv:1610.00620*, 2016.

[3] I. Aekaterinidis and P. Triantafillou, "Pastrystrings: A comprehensive content-based publish/subscribe dht network," in *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, 2006, pp. 23–23.

[4] I. Awan, M. Younas, and W. Naveed, "Modelling qos in iot applications," in *2014 17th International Conference on Network-Based Information Systems*, Sept 2014, pp. 99–105.

[5] R. Barazzutti, T. Heinze, A. Martin, E. Onica, P. Felber, C. Fetzer, Z. Jerzak, M. Pasin, and E. Rivire, "Elastic scaling of a high-throughput content-based publish/subscribe engine," in *2014 IEEE 34th International Conference on Distributed Computing Systems*, June 2014, pp. 567–576.

[6] P. Bellavista, A. Corradi, and A. Reale, "Quality of service in wide scale publish-subscribe systems," *IEEE Communications Surveys & Tutorials*, 2014.

[7] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[8] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16. [Online]. Available: http://doi.acm.org/10.1145/2342509.2342513

[9] G. Bouloukakis, N. Georgantas, A. Kattepur, and V. Issarny, "Timeliness evaluation of intermittent mobile connectivity over pub/sub systems," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, ser. ICPE '17. New York, NY, USA: ACM, 2017, pp. 275–286. [Online]. Available: http://doi.acm.org/10.1145/3030207.3030220

[10] F. Cao and J. P. Singh, "Efficient event routing in content-based publish-subscribe service networks," in *IEEE INFOCOM 2004*, vol. 2, March 2004, pp. 929–940 vol.2.

[11] N. Carvalho, F. Araujo, and L. Rodrigues, "Scalable qos-based event routing in publish-subscribe systems," in *Network Computing and Applications, Fourth IEEE International Symposium on*. IEEE, 2005, pp. 101–108.

[12] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Transactions on Computer Systems (TOCS)*, vol. 19, no. 3, pp. 332–383, 2001.

[13] B. Cheng, A. Papageorgiou, and M. Bauer, "Geelytics: Enabling on-demand edge analytics over scoped data sources," in *2016 IEEE International Congress on Big Data (BigData Congress)*, June 2016, pp. 101–108.

[14] S. P. Curram and J. Mingers, "Neural networks, decision tree induction and discriminant analysis: An empirical comparison," *The Journal of the Operational Research Society*, vol. 45, no. 4, pp. 440–450, 1994. [Online]. Available: http://www.jstor.org/stable/2584215

[15] C. Delimitrou and C. Kozyrakis, "ibench: Quantifying interference for datacenter applications," in *2013 IEEE international symposium on workload characterization (IISWC)*. IEEE, 2013, pp. 23–33.

[16] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, Jun. 2003. [Online]. Available: http://doi.acm.org/10.1145/857076.857078

[17] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84 – 106, 2013, including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X12001318

[18] J. Y. Fernndez-Rodrguez, J. A. lvarez Garca, J. Arias Fisteus, M. R. Luaces, and V. Corcoba Magaa, "Benchmarking real-time vehicle data streaming models for a smart city," *Inf. Syst.*, vol. 72, no. C, pp. 62–76, Dec. 2017. [Online]. Available: https://doi.org/10.1016/j.is.2017.09.002

[19] ——, "Benchmarking real-time vehicle data streaming models for a smart city," *Inf. Syst.*, vol. 72, no. C, pp. 62–76, Dec. 2017. [Online]. Available: https://doi.org/10.1016/j.is.2017.09.002

[20] D. Fesehaye, Y. Gao, K. Nahrstedt, and G. Wang, "Impact of cloudlets on interactive mobile cloud applications," in *2012 IEEE 16th International Enterprise Distributed Object Computing Conference*, Sept 2012, pp. 123–132.

[21] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[22] J. Gascon-Samson, F. P. Garcia, B. Kemme, and J. Kienzle, "Dynamoth: A scalable pub/sub middleware for latency-constrained applications in the cloud," in *2015 IEEE 35th International Conference on Distributed Computing Systems*, June 2015, pp. 486–496.

[23] J. Gascon-Samson, J. Kienzle, and B. Kemme, "Multipub: Latency and cost-aware global-scale cloud publish/subscribe," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 2075–2082.

[24] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[25] S. Guo, K. Karenos, M. Kim, H. Lei, and J. Reason, "Delay-cognizant reliable delivery for publish/subscribe overlay networks," in *2011 31st International Conference on Distributed Computing Systems*, June 2011, pp. 403–412.

[26] D. Happ, N. Karowski, T. Menzel, V. Handziski, and A. Wolisz, "Meeting iot platform requirements with open pub/sub solutions," *Annals of Telecommunications*, vol. 72, no. 1, pp. 41–52, Feb 2017. [Online]. Available: https://doi.org/10.1007/s12243-016-0537-4

[27] K. Intharawijitr, K. Iida, and H. Koga, "Analysis of fog model considering computing and communication latency in 5g cellular networks," in *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, March 2016, pp. 1–4.

[28] Z. Jerzak and H. Ziekow, "The debs 2015 grand challenge," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '15. New York, NY, USA: ACM, 2015, pp. 266–268. [Online]. Available: http://doi.acm.org/10.1145/2675743.2772598

[29] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[30] A. Khanna and R. Anand, "Iot based smart parking system," in *2016 International Conference on Internet of Things and Applications (IOTA)*, Jan 2016, pp. 266–270.

[31] B. Krishnamachari and K. Wright, "The publish-process-subscribe paradigm for the internet of things," 2017.

[32] M. Li, F. Ye, M. Kim, H. Chen, and H. Lei, "A scalable and elastic publish/subscribe service," in *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*. IEEE, 2011, pp. 1254–1265.

[33] S. Li, L. D. Xu, and S. Zhao, "5g internet of things: A survey," *Journal of Industrial Information Integration*, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2452414X18300037

[34] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: ACM, 2015, pp. 450–462. [Online]. Available: http://doi.acm.org/10.1145/2749469.2749475

[35] A. Ng, "Learning curves," https://www.coursera.org/lecture/machine-learning/learning-curves-Kont7.

[36] P. Nguyen and K. Nahrstedt, "Resource management for elastic publish subscribe systems: A performance modeling-based approach," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, June 2016, pp. 561–568.

[37] S. J. Pan, Q. Yang *et al.*, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[38] C. H. Papadimitriou, *Computational Complexity*. Addison-Wesley, 1994.

[39] L. Sanchez, L. Muñoz, J. A. Galache, P. Sotres, J. R. Santana, V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodoridis, and D. Pfisterer, "Smartsantander: Iot experimentation over a smart city testbed," *Comput. Netw.*, vol. 61, pp. 217–238, Mar. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.bjp.2013.12.020

[40] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan 2017.

[41] 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.

[42] J. F. Shortle, J. M. Thompson, D. Gross, and C. M. Harris, *Fundamentals of queueing theory*. John Wiley & Sons, 2018, vol. 399.

[43] A. Shukla, S. Chaturvedi, and Y. Simmhan, "Riotbench: An iot benchmark for distributed stream processing systems," vol. 29, 11 2017.

[44] S. Voulgaris, E. Riviere, A.-M. Kermarrec, M. Van Steen *et al.*, "Sub-2-sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks." in *IPTPS*, 2006.

[45] H. Yang, M. Kim, K. Karenos, F. Ye, and H. Lei, "Message-oriented middleware with qos awareness," in *ICSOC/ServiceWave*, 2009.

[46] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, Feb 2014.

[47] B. Zhang, N. Mor, J. Kolb, D. S. Chan, N. Goyal, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, and J. Kubiatowicz, "The cloud is not enough: Saving iot from the cloud," in *Proceedings of the 7th USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 21–21. [Online]. Available: http://dl.acm.org/citation.cfm?id=2827719.2827740