JADE: Tail-Latency-SLO-Aware Job Scheduling for Sensing-as-a-Service

Stoddard Rosenkrantz, Huiyang Li, Prathyusha Enganti, Zhongwei Li, Lin Sun,
Zhijun Wang, Hao Che, and Hong Jiang
Department of Computer Science and Engineering
University of Texas at Arlington
Arlington, Texas 76019
{*stoddard.rosenkrantz, huiyang.li, zhongwei.li, prathyusha.enganti, lxs5171*}*@mavs.uta.edu*
{*zhijun.wang, hche, hong.jiang*}*@uta.edu*

***Abstract*—As the IoT-Edge-Cloud hierarchy is evolving into a mature ecosystem, large-scale Sensing-as-a-Service (SaS) based services with stringent job service level objectives (SLOs) are expected to emerge as dominant cloud services. A viable business model for SaS must be inherently multi-tier by design and work in a confederated environment involving a large number of voluntary stakeholders who may appear at different tiers. It must also honor privacy and autonomous control of stakeholder resources. This calls for a fully distributed, SLO-aware job resource allocation and scheduling platform to be developed. In this paper, we propose a tail-latency-SLO-aware job resource allocation and scheduling platform for SaS, called JADE. It is a four-tier platform, i.e., cloud, edge cluster, edge, and IoT tiers. To honor the privacy and autonomy of control for individual stakeholders at different tiers, the JADE design follows the design principle of separation of concerns among tiers. Central to its design is to develop a decomposition technique that decomposes SaS service requirements, in particular, the job tail-latency SLO, into task performance budgets for individual sensing tasks mapped to each lower tier. This makes it possible to allow each lower tier to manage its own resources autonomously to meet the sensing task budgets and hence the SaS service requirements, while preserving its privacy and autonomy of control. Finally, preliminary testing results based on both simulation and an initial prototype of JADE are presented to demonstrate the promising prospects of the solution.**

***Keywords*-multi-tier; tail latency-SLO Aware; scheduling; resource allocation;**

I. Introduction

As the IoT-Edge-Cloud hierarchy is evolving into a mature ecosystem in terms of its social and geographical scales, and sensing, computing, and storage capabilities, the cloud is expanding its reach to every corner of the world. This gives rise to the opportunity of developing a whole new category of cloud services, generally known as Sensing-as-a-Service (SaS) [21], [27], [29]. With SaS, a user would have access to sensor data from any part or even the entire world in real time at his/her fingertips. SaS allows users to share a wide variety of sensed data through the IoT-edge-cloud ecosystem. It starts with a user making request to the cloud, which in turn, dispatches sensing tasks to the edge and IoT devices in the areas of interest to *pull* the sensed data up the hierarchy. As a wide variety of sensing data have become available, opportunities arise to develop SaS services of economic, social and political significance, e.g., (in decreasing order of time criticality) earthquake detection and alert [3], object detection and tracking [17], utility monitoring [20], human temperature monitoring for virus detection and prediction [12], and data harvesting for business analytics [19]. To support such applications, especially those with stringent tail-latency SLOs (e.g., earthquake detection and alert and object detection and tracking), however, we argue that a fully distributed, tail-latency-SLO-aware job resource allocation and scheduling platform must be in place first.

First, SaS is in essence a crowdsourcing system [29] involving many loosely coupled stakeholders. For example, a four-layer SaS business model [21] is composed of a sensor-and-sensor-owner layer, a sensor-publisher layer, an extended-service-provider layer, and a user layer. The individual sensor owners in the sensor-and-sensor-owner layer voluntarily subscribe and publish sensing data with one or more publishers in the sensor-publisher layer. An extended service provider in the extended-service-provider layer serves as an agent on behalf of a user in the user layer to make a request for the desired sensing data from different publishers. Clearly, due to privacy concerns, a sensor owner may not be willing to provide the device-level details to its publisher, which in turn, may not be allowed to expose the identities and whereabouts of its subscribers (i.e., sensors) to extended service providers. This means that an extended service provider cannot explicitly allocate sensing resources to meet a user request or job[1] SLO, as it does not have control over or even the knowledge about the sensing resource availability. The existing SLO-aware job scheduling and resource orchestration solutions for both datacenters (e.g., Kubernetes [7], Messos [18], and YARN [1]) and IoT-edge-cloud ecosystem (e.g., [15], [22], [28])

This work was supported by the US NSF under Grant No. CCF XPS-1629625, CCF SHF-1704504 and CCF SHF2008835.

[1]In general, a request may involve multiple rounds of jobs to be executed one at a time. To limit the exposure, in this paper, we only consider single-round requests. In this case, the terms "request" and "job" can be used interchangeably.

978-0-7381-2394-3/20/$31.00 ©2020 IEEE
DOI 10.1109/UCC48980.2020.00058

are simply not up to the task, as they all assume that all the computing/sensing resources can be centrally managed and allocated (see Section II for details).

Second, for large-scale SaS services, a job may spawn up to hundreds of millions of sensing tasks to be dispatched to geographically dispersed edge nodes and/or IoT devices. Whether the job SLO can be met or not is determined by the task response time of the slowest task, a key challenge in job resource allocation and scheduling is to meet the stringent tail-latency SLO for a given service, a de facto SLO for user-facing applications, the task response time budget is a strong function of N_J, i.e., the number of sensing tasks the job spawns, also known as job fanout degree, which may vary significantly from one job to another [26]. For example, as explained in [16], consider a system where the mean task response time is 10ms but with a 99th-percentile latency of one second. Then, for jobs with fanout degree one, only 1% of the jobs will be slower than one second. However, for jobs with fanout degree 100, 63% of the jobs will be slower than one second. Consequently, to meet a given tail-latency SLO, the task response time budget (e.g., in terms of both mean and variance of the task response time) or the task resource demands for tasks belonging to jobs with different job fanout degrees are different! This makes job scheduling extremely challenging, as jobs with different fanout degrees require different task resource allocations. To the best of our knowledge, no existing job resource allocation and scheduling solution is capable of providing job tail-latency-SLO guarantee for jobs with job fanout degree larger than one. To make things worse, for SaS, the job fanout degree, N_J, is unknown at the time a job is initially scheduled. For example, for the aforementioned four-layer business model, upon receiving a user request, an extended service provider will dispatch sensing tasks to a selected number of publishers, denoted as N, to meet the sensing coverage requirement of the user. Each of these publishers, i, for $i = 1, ..., N$, may in turn, dispatch sensing subtasks corresponding to the task received to a selected number of subscribers or sensors for sensing, denoted as n_i, that meets the required sensing coverage associated with that task. Clearly, for the sake of privacy and autonomy of control, n_i is determined by publisher i, meaning that $N_J = \sum_i^N n_i$ is unknown to the job scheduler at the time the job is scheduled at the extended service provider. These two challenges, again, call for a fully distributed solution to be developed.

To tackle the above challenges, this paper takes a first step towards developing a **J**ob-tail-latency-SLO-**A**ware resource allocation and scheduling platform for SaS over IoT-E**D**dg**E**-Cloud Hierarchy (JADE). JADE is a four-tier platform, i.e., cloud, edge cluster, edge, and IoT tiers, in line with the multi-tier SaS business models. In JADE, the cloud (e.g., an extended service provider in the aforementioned four-layer business model) that receives a request from a user (e.g., in the user layer) dispatches sensing tasks of the request to a set of edge clusters (e.g., a set of publishers), each of which in turn, dispatches sensing subtasks of the received task to a set of edge nodes and their respective IoT pools for sensing (e.g., in the sensor-and-sensor-owner layer). To honor the autonomy of control and privacy of individual stakeholders who may appear at different tiers, JADE is fully distributed by design, which strictly follows the design principle of separation of concerns among tiers. Namely, while the cloud tier determines to which edge clusters the sensing tasks need to be dispatched, it is the responsibility of each edge cluster to decide to which edge nodes in the cluster the sensing subtasks need to be further dispatched and how resources need to be allocated at those edge nodes. The core of the JADE design requires the development of an SaS-requirement decomposition technique that allows SaS service requirements, including user request SLO, to be decomposed into task performance budgets at the cloud tier and in turn, a task performance budget for a task into subtask performance budgets for individual subtasks at the edge cluster tier, which finally determines how resources need to be allocated at the edge. This makes it possible to allow lower tiers to manage their own resources autonomously to meet the task/subtask budgets and hence the SaS-requirements, without having to expose their identities and device-level resource details to their upper tiers. Finally, to prove JADE concept and its promising prospects, preliminary testing results based on both simulation and a prototype of JADE are presented.

II. Related Work

First, we note that the existing datacenter job scheduling and resource orchestration solutions that can provide job performance assurance, e.g., Kubernetes [7], Mesos [18], and YARN [1] cannot be adopted as job resource allocation and scheduling platforms for SaS for two reasons.

First, such solutions assume that the entire datacenter is owned by a single stakeholder, i.e., the datacenter service provider. As such, the server resources for all the servers are under the full control of a resource manager (RM). Hence, by working with RM, a job scheduler can spawn tasks for a job to those servers with the right amounts of resources allocated to meet the job performance target. Here we must note that a two-tier variant of YARN, known as YARN federation [**?**], that groups servers in a cluster into sub-clusters appears to match well with the multi-tier SaS business models. In reality, however, they are fundamentally different models. While the YARN federation improves the scalability of YARN by distributing jobs (not tasks) to different sub-clusters, each running a local RM, a multi-tier SaS model requires that the tasks (not jobs) be distributed to different sub-clusters at the next tier. Furthermore, the YARN federation requires that RMs in different sub-clusters exchange sub-cluster resource availability information to

facilitate the possible migration of some of the tasks for the job to some sub-clusters other than the one the job is mapped to. Obviously, the YARN federation does not intend to preserve the privacy and autonomy of control for individual sub-clusters, an essential requirement of the multi-tier SaS business models.

Second, the above datacenter solutions are not capable of providing effective resource allocation for jobs with tail-latency SLO. This is simply because no existing solution is capable of determining the task performance budget or task resource demands for jobs with any given job fanout degree. As a result, to ensure that all the jobs for a given service will meet the tail-latency SLO for that service, the current practice is to overprovision the resources to ensure that the worst-case scenario, i.e., the job with the largest job fanout degree, will meet the tail-latency SLO, resulting in inefficient resource utilization [23].

There are some existing orchestration platforms being proposed for the IoT-Edge-Cloud ecosystem, e.g., [15], [22], [28]. Although mostly multi-tier by design, they all require that the resource availability information from the IoT and edge tiers to be conveyed to the cloud tier for centralized control. They also assume that resource orchestration is under the control of a single stakeholder, at least at the edge and cloud tiers. Moreover, they are not concerned with tail-latency-SLO-aware job scheduling and resource allocation.

We also note that some open-source projects that purposely target IoT and edge computing exist, including two lightweight versions of Kubernetes [7] (i.e., K3S [6] and KubeEdge [5]) and several projects under the Linux Foundation Edge organization (LF Edge) [14] (e.g., Akraino, EdgeX, Fledge, EVE, and Open Horizon). While KubeEdge allows a Kubernetes master running in a cloud to gain control over its worker nodes at the edge, K3S runs in an edge cluster where it orchestrates the resource allocation among all the edge nodes in the cluster. However, as lightweight versions of Kubernetes, both solutions inherit the aforementioned features of Kubernetes, making them unsuitable for SaS. LF Edge "will create a common framework for hardware and software standards and best practices critical to sustaining current and future generations of IoT and edge devices" [13]. Its projects are not targeting a given use case or business model, but rather the building blocks that can be used to enable various use cases and business models and hence, are orthogonal and complementary to JADE. For example, JADE may incorporate EdgeX for standard-based communications between IoT devices and edge nodes; Open Horizon for dynamically adding, deleting or swapping containerized task modules associated with different SaS services at the edge nodes, where the resources are constrained; and/or EVE/K3S for resource orchestration at the edge-cluster tier. Finally, we also note that none of the open source projects has addressed tail-latency-SLO-guaranteed job resource allocation and scheduling.

Next, we also note that commercial IoT-cloud development platforms, e.g., IBM IoT foundation [4], AWS IoT [2], Azure IoT suite [8], provide tools and services that enable developers to integrate their IoT devices into the cloud. However, such platforms are likely to be used by a single stakeholder only who owns all the IoT devices and is tied to a single cloud account.

Finally, many commercial SaS systems exist, e.g., LeanHeat [10], an IoT-based energy monitoring and analysis system for optimized heat distribution to houses and apartments; NetSuite [9], an inventory tracking and alert system; VN Cloud [11], a vending machine management system, just to name a few. However, such systems are proprietary and vertically designed from bottom up for a specific sensing application and stakeholder, making it difficult, if not impossible, for them to be adopted to serve as a common platform involving many stakeholders and shared by different SaS services.

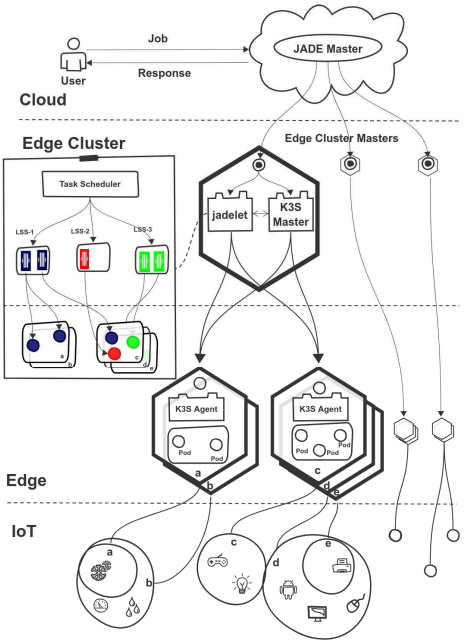


Figure 1: JADE Architecture, highlighted with architectural details for one edge cluster that covers two separate IoT sensing areas

III. JADE

In this paper, we consider both query-based long-run (or long-run in short) and on-demand SaS services with tail-latency SLOs. Enabling a long-run SaS service involves (a) initial resource allocation; and (b) run-time job scheduling and resource scaling to adapt to load changes. For an on-demand SaS service, a user request will trigger both (a) and (b) at run-time. In the following subsections, we first give an overview of JADE, then we discuss the decomposition technique that enables (a) and (b).

A. JADE Overview

JADE adopts a four-tier architecture, as shown in Fig. 1. Each edge node (e.g., a server in a cloudlet) services a pool of IoT devices with possibly more than one type of sensing capability in an area. In turn, a group of edge nodes covering one or more areas forms an edge cluster with a master server (e.g., again a server in a cloudlet). The edge-cluster master manages edge and IoT resources in the edge cluster autonomously without exposing IoT/edge device-level details to the cloud tier. A JADE master running in the cloud tier is mainly responsible for initiating job resource allocation and scheduling for all SaS services running on JADE. The JADE operator (e.g., an extended service provider) owns the JADE master. It may either own or have contractual agreements with the owners of the edge-cluster masters, which in turn, may own or have contractual agreements with the owners of the edge nodes, and so on.

In this paper, we focus on the JADE design in the top three tiers, assuming that the data sensing mechanism is publish-and-subscribe-based, meaning that the sensed data from an IoT pool are periodically pushed to or fetched by the associated edge node and recorded in its data repository. With this mechanism, a sensing subtask arriving at the edge node retrieves sensed data directly from the data repository of the edge node, rather than from the IoT pool.

Each edge node in an edge cluster registers with the edge-cluster master its information, which may include the edge node ID, a list of available types of sensing data it is willing to share (e.g., temperature, humidity, and video) and for each type, a list of sensing features, e.g., in terms of the available degrees of sensing fidelity; sensing frequency; and the area covered. As one can see, the registered information does not include the identity and device-level details (e.g., CPU and memory) of individual IoT devices behind each edge node.

The master in each edge cluster runs two software modules, i.e., a K3S master [6] with the extension to have a northbound and westbound APIs for communication with the JADE master and the other module, i.e., Jadelet, the client-side software of JADE, respectively. The K3S master and K3S agents (i.e., the client-side software of K3S) running in individual edge nodes are responsible for the initial (run-time) pod resource allocation for micro-services that implement subtask workflow for a long-run (on-demand) SaS service and for pod scaling at runtime, upon receiving a scaling request from the Jadelet. Note that as a lightweight version of Kubernetes, K3S [6] is designed for resource orchestration in an edge cluster environment where the computing and memory resources are limited. Without reinventing the wheels, we repurposed it as the resource orchestration solution for each edge cluster in JADE.

The Jadelet module is responsible for processing tasks received from the JADE master and for per-edge-node queuing and dispatching of the subtasks to the edge nodes (see the embedded diagram in Fig. 1). Based on the registration information obtained from its edge nodes, the Jadelet registers and updates with the JADE master in the cloud a summary of its overall sensing capabilities in similar formats as those received from an edge node. Again, the identities and detailed device-level resource availabilities of the edge nodes in the edge cluster are not conveyed to the JADE master.

The JADE master in the cloud tier is mainly responsible for initiating initial resource allocation for long-run services; job scheduling for long-run services; and joint job scheduling and resource allocation for on-demand services.

B. Job Resource Allocation

In this section, we propose an SaS-requirement decomposition technique that decomposes the job resource allocation for both long-run and on-demand services into a two-tier process, one at the cloud tier and the other at the edge-cluster tier. For on-demand services, since job resource allocation must be done jointly with the job scheduling, both of them are presented in this section. A job scheduling solution for long-run services will be presented in the next section.

Cloud Tier: Consider the case where a tenant wants to deploy a long-run service over JADE. The tenant may provide the following information to the JADE operator for initial resource provisioning: $\{\lambda, \{p, x_p\}, S, \{C_k, T_k\}\}$, where λ is the desired query (or job) throughput; $\{p, x_p\}$ pair represents the tail-latency SLO in terms of the pth-percentile job latency of x_p time units; S is the expected average size of the total sensing area per query; $\{C_k, T_k\}$ (for $k = 1, ..., K$) represents a set of K possible configurations of pod resources[2], C_k's, one of which is to be selected and allocated at the edge nodes, and the corresponding measured average subtask execution times, T_k's (as an SaS service developer, the tenant is responsible for providing such information). For a tenant who wants to request for an on-demand service, he/she may submit a job directly to the JADE master with the following information: $\{\{p, x_p\}, S, \{C_k, T_k\}\}$. It differs from that of long-run services in that $\lambda = 0$, since an on-demand service is over as soon as its one-time job is fulfilled, and S defines the exact areas where the sensing data need to be collected.

Upon receiving the above information from a tenant for a long-run (on-demand) service, based on S and the information it has on record regarding the sensing capabilities and areas individual edge clusters cover, the JADE master estimates an expected average number of edge clusters, $\bar{N}$ (an exact number of edge clusters, N), the job will fan out to.

[2]Here a pod is the smallest containerized resource that implements the complete subtask runtime. Multiple pods may be allocated to run subtasks in parallel.

From the JADE master's point of view, the job execution process can be mathematically modeled as a black-box Fork-Join model, with the average job arrival rate, λ ($\lambda = 0$ for on-demand services). Each job spawns $\bar{N}$ (N) tasks to be dispatched to $\bar{N}$ (N) black-box fork nodes (i.e., the edge clusters) to be processed with barrier synchronization, meaning that the job is not finished until the slowest task finishes. In our recent work [23]–[25], we were able to show that for the black-box Fork-Join model, the job tail-latency SLO in terms of $\{p, x_p\}$ can be translated into a task budget pair, $\{E(n_f), V(n_f)\}$, for each of the n_f tasks spawned by the job, which are dispatched to n_f Fork nodes simultaneously to be queued and processed, where $E(n_f)$ and $V(n_f)$ are the maximum allowable mean and variance of the task response time in order to meet the job SLO. So instead of attempting to directly allocate the edge resources, which is not viable at the cloud tier, with $\bar{N}$ (N) and $\{p, x_p\}$, the JADE master estimates the task budget pair, $\{E(\bar{N}), V(\bar{N})\}$ ($\{E(N), V(N)\}$) based on the above result. Furthermore, for long-run services, based on $\bar{N}$ and λ, the JADE master also estimates the expected average task arrival rate, λ_i (note that $\lambda_i = 0$ for on-demand services), at any given edge cluster i. It then sends each edge cluster with a request that includes $\{\{E(\bar{N}), V(\bar{N})\}, S_i, \lambda_i, \{C_k, T_k\}\}$ for the long-run service or a task to each of those N edge clusters together with $\{\{E(N), V(N)\}, S_i, \{C_k, T_k\}\}$ for the on-demand service, where S_i is the expected sub-area in edge cluster i to be covered.

Edge-Cluster Tier: Upon receiving a resource allocation request for the long-run service (a task for the on-demand service), with S_i, the Jadelet in edge cluster i first estimates a fanout degree of the request (task) n_i, i.e., the number of subtasks to be dispatched.

Second, the Jadelet finds the least-cost pod configuration that meets task budget, $\{E(\bar{N}), V(\bar{N})\}$ ($\{E(N), V(N)\}$), as follows. Assume that each of the n_i queues is modeled as an M/M/1 queuing server[3] [31] with average service time T_k for pod of configuration C_k. It is well-known [31] that for this queuing server, the subtask response time distribution $F_{i,k}(t)$ can be exactly expressed as, $F_{i,k}(t) = 1 - e^{-(T_k^{-1} - \lambda_i)t}$ ($\lambda_i = 0$ for on-demand services). Then based on the extreme value theorem [30], the task response time distribution can be approximately (exactly) given as $F_{i,k}^{n_i}$. This allows both $e_{i,k}(n_i)$ and $v_{i,k}(n_i)$, the mean and variance of the task response time for configuration C_k, for $k = 1, 2, ..., K$, to be calculated. Then, the pod configuration C_k is considered feasible if and only if $e_{i,k}(n_i) \leq E(\bar{N})$ ($E(N)$) and $v_{i,k}(n_i) \leq V(\bar{N})$ ($V(N)$). From all the feasible configurations C_k's, the Jadelet will then select the one with the least cost and inform the K3S master to allocate pod resources at the edge nodes accordingly. Upon receiving the acknowledgment from the K3S master, the Jadelet will report back to the JADE master the completion of resource allocation or dispatch the sensing subtasks to those edge nodes, for long-run or on-demand services, respectively. If, on the other hand, no feasible configuration is found or an edge node does not accept any of the feasible configurations, the Jadelet will report the failure of resource allocation to the JADE master. In this case, the JADE master may renegotiate with the tenant on possible relaxation of the SaS-requirements in terms of job throughput, tail-latency SLO, or sensing coverage and then repeat the resource allocation process.

[3]Namely, the FIFO queuing server with Poisson arrival process and exponential service time distribution. This queuing model is expected to be reasonably accurate as long as the pod resources are carefully configured to ensure that the service time is short-tailed and hence can be model by exponential distribution.

C. Job Scheduling for Long-run Services

Again, this is a two-tier scheduling process.

Cloud Tier: The job scheduling process at this tier is the same as the one for on-demand services. Namely, the job may come with the sensing requirements in terms of $\{\{p, x_p\}, S\}$, in addition to possibly other requirements, such as sensing fidelity. With S, the JADE master decides the job fanout degree N and then estimates the task budget, $\{E(N), V(N)\}$, before dispatching tasks to the selected edge clusters.

Edge-Cluster Tier: Similar to the case of on-demand services, upon receiving a sensing task with $\{S_i, \{E(N), V(N)\}$, the Jadelet in edge cluster i first decides n_i, the task fanout degree, based on S_i. Second, the Jadelet dispatches the subtasks to the queues corresponding to those n_i edge nodes. Meanwhile, the Jadelet compares $\{e(n_i), v(n_i)\}$, the most recently measured mean and variance of the task response time for tasks with fanout degree n_i, against $\{E(N), V(N)\}$, to predict whether the edge cluster is likely to be under utilized (e.g., $e(n_i) + v(n_i) << E(N) - V(N)$) or overloaded (e.g., $e(n_i) + v(n_i) > E(N) + V(N)$). If it is, the Jadelet issues a request to its counterpart, the K3S master, for pod resource scaling (an algorithm to do so effectively is yet to be developed). Note that in our solution, JADE does not reject a task even if it may not meet the task budget. This is because given the problem size that JADE has to deal with, it must have a mechanism (to be developed) in place to deal with the unexpected long subtask/task delays (due to, e.g., hardware/software failures) at the edge cluster/cloud tiers, e.g., by setting up timeouts for subtasks/tasks. So, although doing so may cause the violation of SLO for some jobs, it is expected that it will not have a long lasting negative effect on the overall job performance.

Finally, we note that the above solution requires that the Jadelet continuously measures and updates the mean,

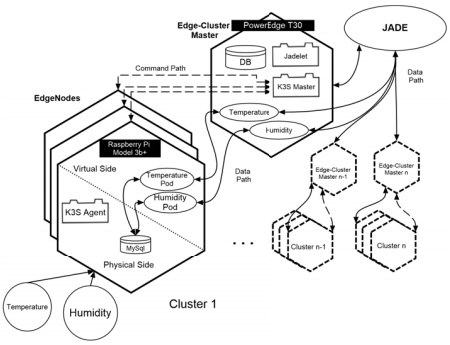


Figure 2: The design diagram for JADE, highlighted with the details of one edge cluster

Figure 3: JADE Prototype

$e(n_i)$, and variance, $v(n_i)$, of the task response time for all possible task fanout degrees n_i's. Since a task response time can be easily estimated by taking the different between the measured task completion time and the measured task fanout time, which are available from the log files, the overhead for doing so should be easily manageable.

IV. Prototyping

In this section, we describe an initial implementation of JADE. Fig. 2 depicts the design. We consider two types of long-run SaS services that allow users to query temperature and humidity of the sensed areas, respectively. One temperature sensor and one humidity sensor are physically connected to a Raspberry Pi Model 3b+, a single board computer, forming a pool of two IoT sensing devices connecting to a Raspberry Pi edge node. Multiple replicas of this IoT-edge node are created to enable an edge cluster. The edge-cluster master, including the K3S master and Jadelet with per-SaS-service and per-edge-node FIFO queues, runs in Dell PowerEdge T30. Accordingly, the measurement-based mechanism for task budget estimation proposed in Section is implemented.

K3S successfully allocates the pods for the two SaS services in the edge nodes through the control path given in Fig. 2. Note that while the control path is for resource allocation, the data path is for job scheduling.

Multiple replicas of the above edge cluster with pre-loaded pods are then generated to form a multi-edge-cluster-based JADE platform, shown in Fig. 3. The current prototype only allows the JADE master to perform job scheduling through the data path without run-time resource scaling.

Since the sensors have no native computing, storage, or power, we configure the host (Physical Side in Fig. 2) system in Raspberry Pi to poll the sensors every 5 minutes and store the data in its local storage. The following are some examples of the stored raw temperature data:

```
[{"TIME":"2020-02-19 00:00:39","Temperature":21.7},
{"TIME":"2020-02-19 00:01:06","Temperature":21.6},
{"TIME":"2020-02-19 00:05:39","Temperature":21.6},
...
{"TIME":"2020-02-19 15:13:19","Temperature":21.6}]
```

In turn, the sensing task from the pod in the edge node fetches the raw data from the storage, which are then processed in the pod. Below is an example of the processed data,

```
{"data":{"Temperature":{"Max":21.7,"Min":20,
"Avg":20.85300751879699},"Size":266}}
```

Note that the data may not contain entire records but only fields needed to satisfy the subtask. The request itself may also be designed to select a specific window or range of data.

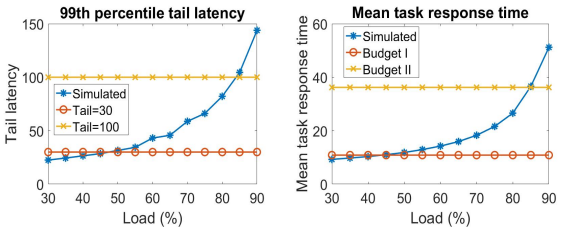
Finally, the processed data is sent back to the corresponding pod in the edge-cluster master. Upon receiving the processed data from both edge nodes, the pod forwards the data up to the cloud without further processing.

V. Testing

In this section, we test by simulation whether JADE can indeed provide job tail-latency SLO guarantee or not.

Consider JADE with 100 edge clusters and 1000 edge nodes per edge cluster and jobs for a given long-run SaS service with fixed fanout degrees of N=100 and n_i=1000 for jobs and tasks, respectively. Further assume that the subtask service time that includes the sensing data retrieval and processing times in an edge node follows an exponential distribution with mean service time of 1 time unit (the unit can be millisecond, second, and so on). Also assume that the job flow follows a Poisson arrival process, which implies that all the subtask queues are M/M/1 queuing servers.

We consider two different tail-latency SLOs, i.e., the 99th-percentile job response times of 30 and 100 time units for cases I and II, respectively. It can be shown that with the M/M/1 queuing for subtasks in an edge cluster, the ratio of the mean and variance of task response time is approximately a constant, independent of the load, or the job arrival rate, λ. In this case, according to [23], whether a tail-latency SLO can be met is uniquely determined by $E(N)$, i.e., the budget for the mean task response time, which takes values of 10.86 (Budget I) and 28.95 (Budget II) time units for cases I and II, respectively. It means that if the mean task response time for case I (II) is no more than 10.86 (28.95) time units, the 99th percentile tail latency should be no more than 30 (100) time units.



(a) (b)

Figure 4: (a) Simulated tail latency and the tail latency SLOs of 30 and 100 time units; and (b) Simulated mean job response time and mean job response time budgets of 10.86 and 28.95 time units

Figure 4 gives the simulation results. For case I, one can see from Figure 4 (b) that the simulated mean task response time is within budget I when the load is smaller than or equal to 45%. The corresponding results in Figure 4 (a) shows that the simulated tail latency meets the tail latency SLO at the load of 47% or lower. Similarly, in case II, the simulated mean job response time is within budget II at the load of 82% or lower and the corresponding tail latency meets the tail latency SLO of 100 time units at the load of 84% or lower. Although a comprehensive evaluation is needed, these results do suggest that the decomposition technique can indeed provide the tail-latency SLO guarantee with possibly a small percentage of resource overprovisioning, e.g., only 2% for both cases studied.

VI. Conclusions

This paper proposes JADE, a job resource allocation and scheduling platform that supports job-SLO-guaranteed Sensing-as-a-Service (SaS) over IoT-Edge-Cloud hierarchy. It is a highly scalable, four-tier distributed solution. At the core of JADE is the development of a decomposition technique to allow SaS requirements including job SLO to be translated into task/subtask performance budgets at the edge-cluster/edge tiers. This makes it possible to allow each lower tier to manage its own resources autonomously to meet the sensing task budgets and hence the SaS requirements, while preserving its privacy and autonomy of control. Finally, preliminary testing results based on both simulation and an initial prototype of JADE are presented to demonstrate the promising prospects of the solution.

REFERENCES

[1] Apache Hadoop YARN. https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/.

[2] AWS IoT. https://aws.amazon.com/iot/.

[3] Earthquake Hazards. https://www.usgs.gov/natural-hazards/earthquake-hazards/education.

[4] IBM Watson IoT Platform . https://internetofthings.ibmcloud.com/.

[5] KubeEdge. https://kubeedge.io/.

[6] Kubernetes (K3s). https://k3s.io/.

[7] Kubernetes (K8s). https://kubernetes.io/.

[8] Microsoft Azure. https://azure.microsoft.com/.

[9] Oracle Netsuite. https://www.netsuite.com/portal/home.shtml.

[10] Smart building control and maintenance powered by IoT. https://leanheat.com/.

[11] VendNovation. https://www.vendnovation.com/vn-cloud.

[12] virus. https://www.novuslight.com/industrial-thermal-cameras-for-pandemic-fever-detection_N10300.html.

[13] Linux foundation edge. In *https://www.lfedge.org*, 2019.

[14] Linux foundation edge projects. In *https://www.lfedge.org/projects*, 2019.

[15] M. Alarbi and H. Lutfiyya. Sensing as a service middleware archecture. In *Proceedings of the IEEE 6th International Conference on Future Internet of Things and Cloud*, pages 399–406, 2018.

[16] J. Dean and L. A. Barroso. The Tail at Scale. *Communications of the ACM*, 56(2):74–80, Feb. 2013.

[17] J. Gall, A. Yao, N. Razavi, L. V. Gool, and V. Lempitsky. Hough forests for object detection, tracking, and action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:2188–2202, 2011.

[18] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 295–308, 2011.

[19] J. Jin, J. Gubbi, S. Marusic, and M. Palaniswami. Harvesting big data to enhance supply chain innovation capabilities: An analytic infrastructure based on deduction graph. *International Journal of Production Economics*, 165:223–233, 2015.

[20] R. Kumar, K. Khatri, M. Imran, and H. A. Khattak. Towards smart utility monitoring and management. 2019.

[21] Namal, Suneth and Gamaarachchi, and MyoungLee, Gyu and Um, Tai-Won . Innovations in NGN: Future Network and Services, K-INGN, ITU-T Kaleidoscope Academic Conference. In *Wiley Transactions on Emerging Telecommunications Technologies*, 2015.

[22] S. Nastic, H.-L. Truong, and S. Dustdar. A Middleware Infrastructure for Utility-based Provisioning of IoT Cloud Systems. In *Proceedings of the IEEE/ACM Symposium on Edge Computing*, pages 28–40, 2016.

[23] M. Nguyen, S. Alesawi, N. Li, H. Che, and H. Jiang. Forktail: A black-box fork-join tail latency prediction model for user-facing datacenter workloads. In *Proceedings of the International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2018.

[24] M. Nguyen, S. Alesawi, N. Li, H. Che, and H. Jiang. A blackbo-box fock-join latency prediction model for data-intensive applications. In *IEEE Transactions on Parallel and Distributed Systems*, to appear.

[25] M. Nguyen, Z. Li, D. Feng, H. Che, J. Lei, and H. Jiang. The tail at scale: How to predict it? In *the 8th USENIX Workshop on Hot Topics in Cloud Computing*, July 2016.

[26] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani. Scaling Memcache at Facebook. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation - NSDI'13*, pages 385–398, Apr. 2013.

[27] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Sensing as a service model for smart cities supported by Internet of Things. *Wiley Transactions on Emerging Telecommunications Technologies*, 25(1):81–93, 2013.

[28] D. Santoro, D. Zozin, D. Pizzolli, F. Pellegrini, and S. Cretti. Foggy: A Platform for Workload Orchestration in Fog Comuting Environment. In *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science*, 2017.

[29] X. Sheng, J. Tang, X. Xiao, and X. Guoliang. Sensing as a service: Challenges, solutions and future directions. volume 13, 2013.

[30] Wikipedia. Extreme value theorem. In *https://en.wikipedia.org/wiki/Extreme_value_theorem*, 2019.

[31] Wikipedia. M/m/1 queue. In *https://en.wikipedia.org/wiki/M/M/1_queue*, 2020.