# BRACELET: Hierarchical Edge-Cloud Microservice Infrastructure for Scientific Instruments' Lifetime Connectivity

Phuong Nguyen, Steven Konstanty, Tarek Elgamal, Todd Nicholson, Stuart Turner, Patrick Su, Klara Nahrstedt,
Timothy Spila, Roy H. Campbell, John Dallesasse, Michael Chan, Kenton McHenry
*University of Illinois at Urbana-Champaign*

***Abstract*—Recent advances in cyber-infrastructure have enabled digital data sharing and ubiquitous network connectivity between scientific instruments and cloud-based storage infrastructure for uploading, storing, curating, and correlating of large amounts of materials and semiconductor fabrication data and metadata. However, there is still a significant number of scientific instruments running on old operating systems that are taken offline and cannot connect to the cloud infrastructure, due to security and performance concerns. In this paper, we propose BRACELET - an edge-cloud infrastructure that augments the existing cloud-based infrastructure with edge devices and helps to tackle the unique performance & security challenges that scientific instruments face when they are connected to the cloud through public network. With BRACELET, we put a networked edge device, called cloudlet, in between the scientific instruments and the cloud as the middle tier of a three-tier hierarchy. The cloudlet will shape and protect the data traffic from scientific instruments to the cloud, and will play a foundational role in keeping the instruments connected throughout its lifetime, and continuously providing the otherwise missing performance and security features for the instrument as its operating system ages.**

## I. INTRODUCTION

With the proliferation of digital technologies, instrumentation, and pervasive networks for data collection, sharing, and analysis, there are increasing needs for advanced cyber-infrastructure to support data-driven and interdisciplinary scientific research. However, related efforts [1] mainly focus on homogenous, well-organized data in an offline or batch manner (e.g., in astronomy and high energy physics), and much less effort has been on long-tail data, i.e., data of small or medium sizes collected during day-to-day research, or "dark data", i.e., unpublished data including results from failed experiments and records viewed as ancillary to published studies. Therefore, in material sciences for example, it often takes a long time from the discovery of new materials to fabrication of new and next-generation devices based on the new materials [2].

In order to speed up new discoveries, there have been recent efforts [3, 4] that focus on enabling digital data sharing and ubiquitous network connectivity between scientific instruments (e.g., SEMs, TEMs, AFMs) and cloud-based storage infrastructure for uploading, storage, curation, and correlation of large amounts of materials and semiconductor fabrication data and metadata. However, there is still a significant number of scientific instruments that run their scientific software tools on old operating systems (e.g., Windows XP, Windows NT, Windows 2000). Since these OSes cannot operate at the network speed of a powerful cloud and are not patched with the latest security patches, the instruments are taken offline and cannot connect to the cloud infrastructure. This is because if these instruments were put on the network, they would be destroyed by viruses and might represent major security threats and performance bottlenecks to the very expensive instruments and the overall network infrastructure. Furthermore, this situation will not go away, since instrument companies do not upgrade their instrument software at the same frequency with which the computing companies upgrade their OSes[1]. Even more recent OSs, such as Windows 7, will become obsolete in the near future, and scientific instruments running on Windows 7 will eventually join the group of offline instruments. As a result, the current networked solution for scientific instruments is not evolvable and represents a major barrier to accelerating the pace of discovery and deployment of advanced cyber-infrastructure.

In order to bridge the security and performance gaps between disconnected scientific instruments and cloud-based cyberinfrastructure, in this paper, we propose BRACELET, an edge-cloud infrastructure that introduces networked edge device, called *cloudlet*, in between scientific instruments and cloud as the middle tier of a three-tier hierarchy. The introduction of cloudlets poses several challenges. *First*, cloudlets need to be integrated seamlessly with the existing cloud-based cyberinfrastructure to support offloading and processing of scientific data across edge and cloud. *Second*, since scientific data workloads are often represented by a workflow model with complex interactions and dependencies between tasks, it is required to develop new offloading algorithm for hierarchical edge-cloud infrastructure for workflow-based scientific workload. *Third*, recent survey [5] has shown highly dynamic characteristic of data being uploaded from scientific instruments. Hence, it is important that the edge-cloud infrastructure is able to dynamically adapt its resources to meet certain performance guarantee under limited capacity on edges and cloud. We elaborate

[1]It is often that the instrument companies (e.g., GE, Siemens) stop augmenting/updating their scientific softwares when OSes are upgraded to newer versions or when new OS patches come up. Hence, to make use of the instruments, scientific users have to run the instruments on older OSes.

more on these challenges in Section II.

To address the above challenges, we design BRACELET by extending the state-of-the-art cloud-based microservice cyberinfrastructure [3, 4] to the edges. Specifically, we design an edge-cloud microservice execution and coordination mechanism to support executing scientific workflows seamlessly across edge and cloud. In addition, we propose a novel resource management mechanism to tackle *both* computation placement (i.e., *where* to run a task) of scientific workflow workload and resource allocation (i.e., *how much* resource to be allocated for a task) challenges of the edge-cloud microservice infrastructure. Our resource management mechanism is based on a microservice performance model that is trained from historical performance data and is used to make predictions on near-future performance of microservices. These predictions not only can be used to make decisions on resource allocation of microservices, but also to explore different computation placement options of tasks in workflows to choose the placement scheme that minimizes expected processing delays of workflows.

We have validated the design of BRACELET on real edge-cloud environment of campus cyberinfrastructure and evaluated the proposed resource management mechanism on materials-related scientific data workload. Beside extensive performance design, we also implement a robust security mechanism into BRACELET to ensure that BRACELET is able to protect vulnerable scientific instruments from external threats.

In summary, our contributions are as follow:

- We present *the first* edge-cloud microservice cyberinfrastructure that tackles both performance and security challenges of scientific instruments' lifetime connectivity.
- We design and validate a novel resource management mechanism that is based on a microservice performance model, and tackles both computation placement and resource allocation of microservices.
- We have implemented and deployed the system in a real environment with encouraging evaluation results.

The remaining of the paper is organized as follows. We first provide some background information about our targeted environment in Section II. Then, we give an overview of BRACELET's architecture and describe in details the edge-cloud execution model in Section III. In Section IV, we provide some assumptions and modeling notation for our resource management mechanism presented in Section V. After that, in Section VI, we will present in details our system implementation and BRACELET' security component design that helps to protect vulnerable scientific instrument's computer from external threats. We present our evaluation results in Section VII. We summarize the related work in Section VIII before concluding the paper in Section IX.

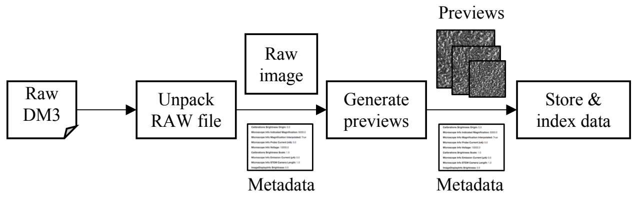


Figure 1: A sample of simple workflow to process raw DM3 file generated from digital microscope.

## II. BACKGROUND

To better understand the target environment of material-related instruments, we provide in this section some background information on the current state of cyberinfrastructure in material-related environment, types of data generated from the instruments, and example of workflows used to process that kind of data.

State-of-the-art cyberinfrastructure in material-related environment [3] uses a two-tier architecture that connects modern scientific instruments directly to a cloud-based infrastructure to support capturing data from instrument, transferring, and processing the digital data in real-time and in trusted manner before archiving, further analysis, visualization for more efficient interpretation and sharing of the experimental results. Data generated from instrument is often in raw format that it requires additional data processing to extract useful information. With material-related data (and in scientific data in general), the data processing step often involves executing a workflow (i.e., in form of a directed acyclic graph) of tasks on the raw data, in which, each task performs a specific processing on the data. Figure 1 shows an example of a data processing workflow of three tasks that involves in processing raw DM3 file generated from digital microscope. In the first task, the raw file is unpacked into image part and metadata. Next, the raw image is processed and multiple previews of different sizes are generated. In the last task, the final data is stored into database and metadata is indexed to make it searchable. All tasks are performed on the cloud-based infrastructure.

To support processing heterogeneous types of data uploaded from instruments, latest data cyberinfrastructures [3, 4] leverage a microservice-based design, in which each data processing task is model as a microservice with independent request queue and a set of task consumers to handle requests (more on this in Section III). This microservice-based approach has demonstrated its effectiveness in handling complex workflows under dynamic workfload situations.

However, as motivated in Section I, since a large number of scientific instruments are still offline, it is intuitive to introduce an intermediate mean, or edge device, to help to connect these instruments to the cloud-based infrastructure. The introduction of the edge devices not only helps to protect vulnerable instruments from security threats but also opens opportunities for off-loading computation from the cloud-based cyberinfrastructure, which is often limited in resource capacity, to the edges. In addition, as recently shown in [6], the hierarchical edge-cloud architecture is also more efficient in handling peak workloads, compared to a flat edge-cloud architecture that does not take into account the hierarchy structure. However, supporting computation offloading across hierachical edge-cloud cyberinfrastructure to serve dynamic scientific workloads presents a number of challenges:

- ***Edge-cloud microservice execution challenge:*** We need to extend the cloud-based microservice execution environment to the edges to allow running microservices at the edges and to support seamless coordination between dependent tasks of a workflow across edge and cloud. We show how to address this challenge in Section III.
- ***Computation placement challenge:*** Since scientific workloads have a workflow model, computation offloading algorithm has take into account the complex dependencies between tasks. In addition, the algorithm also needs to appropriately balance between the benefit of reducing workload at the cloud by running tasks on edges and communication delays caused by transmitting intermediate data between edge-based and cloud-based tasks.
- ***Resource allocation challenge:*** Allocating resources to microservices across edges and cloud becomes even more challenging in cyberinfrastructure environment, which is often deployed on private or on-premise cloud infrastructure with limited resource capacity on both cloud and edges.

In Section V, we present our solution to tackle both computation placement and resource allocation challenges of microservices over hierarchical edge-cloud architecture.

## III. BRACELET ARCHITECTURE

An overview of BRACELET's 3-tier architecture is presented in Figure 2. In particular, the first tier, i.e., *instrument tier*, includes scientific instruments attached to computers running old operating systems that could not directly connect to the cloud (the new instruments that run with more advanced operating systems can connect directly to the cloud in the existing 2-tier architecture). On each instrument's computer, users use a uploader client to upload experiment data upstream. The second tier, i.e., the *edge* or *cloudlet* tier[2], includes edge-based devices, or cloudlets, that consist of two network interfaces: one connects to instruments' VLAN and another connects to the cloud via public network. Lastly, on the third tier, i.e., *cloud tier*, we deploy a cloud-based infrastructure that connects to the public network.

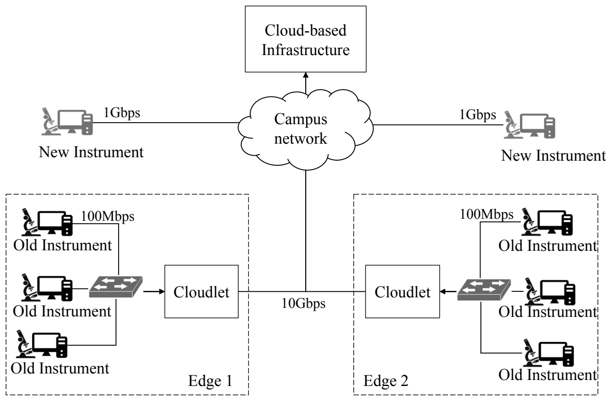


Figure 2: Overview of BRACELET system.

The cloud-based tier supports data processing, curation, storage, correlation, and search of scientific experiment data uploaded from instruments via cloudlets.

### A. BRACELET*'s Micro-service Architecture*

Figure 3 shows the detailed microservice architecture of BRACELET and its performance components. To enable seamless integration of cloudlets to the existing 2-tier cloud-based infrastructure, we design BRACELET by extending the cloud-based microservice architecture [3, 4] to the edges. In particular, while the cloud-based infrastructure operates on the full 5-layer architecture, the cloudlets operates on three layers to enable computational offloading of tasks to the edges and seamless communication between edge- and cloud-based components. In the following, we describe all the layers in details.

*1) Infrastructure Layer:* Infrastructure layer provides a level of abstraction and virtualization of all computation and storage resources across cloud and edges. We leverage container technology for virtualization and use a container orchestration engine to manage the container allocation across edge-cloud infrastructure.

*2) Execution Layer:* We design execution layer using a microservice workflow execution model across cloud and edges. Experiment data uploaded from instruments will be handled by a specific type of *data processing workflow*, each workflow type corresponds to a directed acyclic graph (DAG) of a subset (or all) types of *data processing tasks* that system supports. We model each task as a *microservice*[3] with its own *request queue* that stores the task's requests, and a set of *task consumers* that subscribe to the request queue to perform actual processing of the task's requests.

The communication between dependent tasks in a workflow follows the *publish-subscribe mechanism*. When a task request arrives at the queue, a task consumer subscribing to the queue will pick up the request to process it. After processing the request, the consumer asks the coordination layer (to be described shortly) about the subsequent tasks of

[2]From now, we will use *edge*, *edge device* and *cloudlet* interchangably.

[3]From now, we will use *task* and *microservice* interchangably.

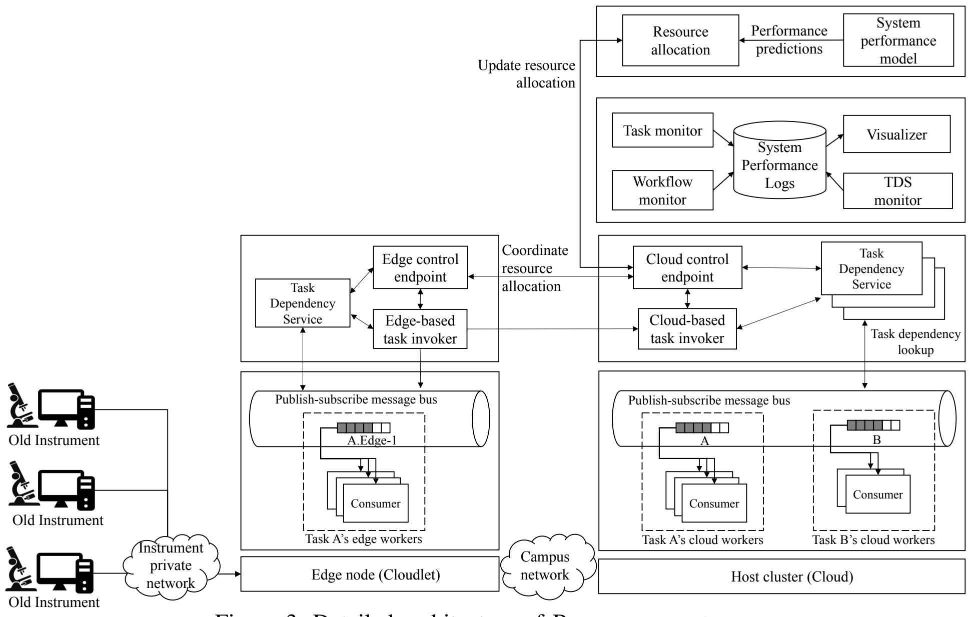


Figure 3: Detailed architecture of BRACELET system.

| Job type | From | To |
|---|---|---|
| Wf1 | Start | A |
| Wf1 | A | B |
| Wf1 | B | C |
| Wf1 | C | End |
| Wf2 | Start | C |
| Wf2 | C | D |
| Wf2 | D | End |
| ... | ... | ... |

Table I: Example of task dependency table.

the workflow and publish the request to the corresponding queues of the subsequent tasks. We assume that all workflow data and intermediate results between tasks are stored in a shared storage system that can be accessed by all microservices across cloud and edges.

A microservice can be deployed on a cloudlet (or multiple cloudlets), on cloud, or on both cloud and cloudlets. The publish-subscribe message bus is available across cloud and edges to enable seamless communication between edge- and cloud-based microservices.

*3) Coordination layer:* On top of the execution layer is the coordination layer that consists of a *task dependency service*, or TDS, that maintains the dependencies between tasks of a workflow (i.e., task dependencies are essentially the directed edges of workflow's task graph) and responds to task dependency lookups from the execution layer. Table I shows an example of task dependencies maintained by TDS for two types of workflows (i.e., `Wf1` and `Wf2`) and 4 types of tasks (i.e., `A`, `B`, `C`, and `D` - please note that the same task can be used by multiple workflow types).

The separation of task coordination from the execution of tasks enables more flexible and scalable workflow composition (i.e., we can support new workflow types by simply creating new set of task dependencies between the existing tasks). To offer high availability and high performance, we designed TDS as an ensemble of multiple TDS instances running on both cloud and edges and maintain a replica of task dependency data on each instance.

To coordinate resource allocation across cloud and edges, coordination layer maintains a *control endpoint* on each cloud and edge side. The *cloud control endpoint* is the centralized entity that receives new resource allocation from the adaptation layer (to be described shortly) and informs other *edge control endpoints* to implement new allocation.

*4) Monitoring layer:* Monitoring layer captures performance metrics (to be described in Section IV) of workflows, microservices, and TDS. These metrics are stored in a performance logs database. Performance data is used by adaptation layer (to be described shortly) to make resource allocation decisions. Although monitoring services are running on the cloud, they still can seamlessly communicate with components running on edges to collect the performance metrics, thanks to the deployment of coordination and execution layers across cloud and edges.

*5) Adaptation layer:* Adaptation layer is the brain of BRACELET system. This layer consists of a *system performance model* that is trained on the performance logs collected by monitoring layer and provides near future performance predictions to help *resource allocation* module to dynamically allocate resources for microservices across cloud and edges. We describe adaptation layer in details in Section V.

| Job type | From | To |
|---|---|---|
| ... | ... | ... |
| Wf1.E1 | Start | A.E1 |
| Wf1.E1 | A.E1 | B.E1 |
| Wf1.E1 | B.E1 | C |
| Wf1.E1 | C | End |
| ... | ... | ... |

Table II: Updated task dependency table with an edge-based version of workflow `Wf1`.

## B. Edge Cloud Microservice Execution Model

We end the architecture section by describing how microservices are initially deployed and how workflows are executed seamlessly across cloud and edges by leveraging the dynamic configuration of workflow's task dependencies.

Since data can be uploaded to the cloud either via a cloudlet or directly from advanced instruments (which are able to connect directly to the cloud without cloudlet), all microservices have to be deployed on the cloud, so that they can be ready to support processing all types of workflows. For each cloudlet, depending on the types of data that is uploaded from instruments to the cloudlet, microservices of the corresponding data processing workflows have to be deployed on the cloudlet. Therefore, the initial deployment of microservices on cloud and cloudlets can be decided in advance with knowledge of the types of uploaded data[4]. For example, if the system supports all types of workflows in Table I, then microservices of all tasks `A`, `B`, `C`, `D` are deployed on the cloud. If only data corresponding to workflow `Wf1` is uploaded through an edge named `E1`, then initial deployment on `E1` will include microservices of the tasks in `Wf1`, namely `A.E1`, `B.E1`, and `C.E1` (i.e., the edge-specific suffix is used to differentiate with cloud-based deployments of `A`, `B`, and `C`).

With the above initial deployment, the execution of a workflow across cloud and edge can be conveniently handled by the cloud control endpoint via dynamic configuration of task dependencies on TDS. For example, to execute a workflow `Wf1` across edge `E1` and cloud (e.g., processing requests of task `A` and `B` on `E1`, and of task `C` on the cloud), the cloud control endpoint simply creates a new edge-based workflow type on TDS, namely `Wf1.E1` (Table II), that directs requests of task `A` and `B` to their `E1`-based microservice deployments, namely `A.E1` and `B.E1` (task `C` is still handled by its cloud-based microservice deployment). After creating the new workflow type `Wf1.E1`, cloud control endpoint will inform edge control endpoint at `E1` to use `Wf1.E1` as the workflow type to process all requests for `Wf1` of data being uploaded via `E1` (instead of the initial cloud-only version of workflow `Wf1` as shown in Table I).

## IV. Models

Before presenting BRACELET's resource management mechanism, we describe microservice resource model, as well as the key performance metrics used by the system.

### A. Microservice Resource Model

Let us consider a system hierarchy with a single centralized cloud and $E$ cloudlets. System supports processing $N$ workflow types that are composed of $J$ types of tasks (i.e., each workflow type corresponds to a DAG of a subset, or all, of $J$ types of tasks). We model each task type $j$ ($1 \leq j \leq J$) as a microservice (cf. Section III) with a request queue and $m_j$ task consumers. Since a task can be deployed both on edge and cloud, we distinguish between cloud-based and edge-based micro-services using superscript $m_j^e$, $0 \leq e \leq E$, with $e = 0$ being cloud-based microservice and $1 \leq e \leq E$ being one of the $E$ edge-based micro-services. To simplify the resource model, we assume that task consumers need uniform computational resource, and thus, system resources can be represented as the numbers of consumers over tasks $\mathbf{m} = \{m_j^e\}$ ($0 \leq e \leq E$, $1 \leq j \leq J$). To capture the changes of number of consumers $\mathbf{m}$ over time, we denote $\mathbf{m}(k)$ as the numbers of consumers over tasks during time window $(T_k, T_{k+1})$: $\mathbf{m}(k) = \{m_j^e(k)\}$, where $m_j^e(k)$ is the number of consumers of task type $j$ located at edge $e$ (with edge 0-th being the cloud) during the $k$-th time window.

### B. Performance Metrics

At workflow level, we are measuring the *average processing time* (or average delay) of each workflow type $i$ ($1 \leq i \leq N$), as well as the average delay over all types of workflows. The *processing delay* of a workflow request is defined as the duration between its arrival time $t$ and the time when the workflow's last task is finished. The average delay of workflow type $i$ over the time window $(T_k, T_{k+1})$, denoted as $d_i(k)$, is calculated by averaging delays of all requests of type $i$ that arrive during $(T_k, T_{k+1})$. We denote $\mathbf{d}(k)$ as the vector form of the set of all average delays of workflow types in the $k$-th time window: $\mathbf{d}(k) = (d_1(k), d_2(k), ..., d_N(k))$. The average delay of requests over all types of workflows in the time window $(T_k, T_{k+1})$ is denoted as $\bar{\mathbf{d}}(k)$.

At task level, the processing delay of a workflow request when it is processed by a microservice is measured from the time the request arrives at task's request queue until the request departs the microservice after being processed by one of the task consumers. As a result, the processing delay includes *both* the waiting time in the queue and the actual processing time by task consumer. Since, according to the Little's law[5], this processing delay is proportional to the number of requests in the microservice (i.e., including requests waiting in the queue and requests being processed by task consumers), or *the number of work-in-progress* (work-in-progress or WIP for short) . Hence, we use work-in-progress to measure the performance of individual microservice. The more work-in-progress a microservice has,

[4]This is a reasonable assumption since the type of uploaded data is specific to the type of instrument, which is known information.

[5]Wikipedia: https://en.wikipedia.org/wiki/Little%27s_law

the longer delay is to be expected. We denote $w_j^e(k)$ as work-in-progress of task $j$ ($1 \leq j \leq J$) on edge $e$ ($0 \leq e \leq E$) during time window $(T_k, T_{k+1})$. We use $\mathbf{w}(k)$ as the vector representation of work-in-progress over all micro-services during $k$-th time window.

## V. BRACELET'S RESOURCE MANAGEMENT

BRACELET' system performance is controlled by allocating resources to micro-services and by placing task computation across edges and cloud. In particular, for each microservice, the more consumers subscribe to a task's request queue, the more requests can be processed in parallel and the less time requests must wait in the queue. Therefore, $\mathbf{m}(k)$ influences the work-in-progress $\mathbf{w}(k)$ and workflow's processing times $\mathbf{d}(k)$. In addition, the flexible execution model of workflows across edge and cloud (presented in Section III-B) enables BRACELET's resource management to make timely decisions on whether to place the computation of a workflow's task on an edge- or cloud-based microservice to balance the workload across the infrastructure.

In this paper, we propose a novel approach to tackle both resource allocation and computation placement challenges of micro-services. In the following sections, we first present a microservice performance model that provides performance predictions of *individual micro-services*. These predictions not only can be used to estimate expected delays of different types of workflows (by aggregating delays of individual micro-services), but also can be used to explore different computation placement options of micro-services and choose the one that minimizes expected processing delays. After introducing the microservice performance model, we will show how to apply the model to solve the resource allocation and computation placement challenges.

### A. Microservice Performance Model

Modeling performance of a system is basically to derive a function that takes system's resource configurations as inputs and produces prediction of system performance in the near future. In this paper, we use artificial neural network, a black-box and data-driven approach with proven approximation power and successful applications in modeling performance of non-linear and complex systems, to model the performance of micro-services.

Specifically, the neural network model takes input $\mathbf{x}$ as the combination of microservice performance output (i.e., $\mathbf{w}(k)$) and microservice's resource configurations (i.e., $\mathbf{m}(k)$) in the current time window $(T_k, T_{k+1})$, and predicts microservice performance in the next time window $(T_{k+1}, T_{k+2})$: $\mathbf{w}(k+1)$. The neural network model consists of $n$ layers, with the number of neurals in each layer is denoted as $S^i$ ($1 \leq i \leq n$). Correspondingly, $\mathbf{W}^i, \mathbf{b}^i$ ($1 \leq i \leq n$) represent the weight matrix and bias of layer $i$-th. Each layer $i$-th also includes a non-linear (except the last layer that uses the linear identity function) activation function $\boldsymbol{f}^i$ to introduce the non-linearity into the network model. The neural network model can be described as a function $\boldsymbol{f}$ of $\mathbf{w}(k)$ and $\mathbf{m}(k)$ via a series of matrix calculations as follows:

$$\begin{aligned}
\mathbf{Z}^1 &= \boldsymbol{f}^1(\mathbf{W}^1(\mathbf{w}(k) \parallel \mathbf{m}(k))^T + \mathbf{b}^1) \\
\mathbf{Z}^2 &= \boldsymbol{f}^2(\mathbf{W}^2\mathbf{Z}^1) + \mathbf{b}^2) \\
&\ldots \\
\mathbf{Z}^n &= \boldsymbol{f}^n(\mathbf{W}^n\mathbf{Z}^{n-1}) + \mathbf{b}^n) \\
\mathbf{w}(k+1) &= \boldsymbol{f}(\mathbf{w}(k), \mathbf{m}(k)) = \mathbf{Z}^n
\end{aligned} \tag{1}$$

To train the microservice performance model, we define a loss function using standard root mean square error to capture the differences between values of work-in-progress predicted by the model (i.e., $w_j^e(k+1)$) and the values actually observed (i.e., $\hat{w}_j^e(k+1)$) over all micro-services on cloud and edges:

$$L(k+1) = \sqrt{\frac{1}{\mathbb{N}} \sum_{e,j} (w_j^e(k+1) - \hat{w}_j^e(k+1))^2} \tag{2}$$

where $\mathbb{N}$ is the total number of micro-services on cloud and edges. The model is trained using gradient descent optimizer and backpropagation is used as the gradient computing technique.

### B. Microservice Work-in-progress Optimization

We formulate the microservice resource allocation problem as an work-in-progress optimization problem (Problem (3)) whose objective is to minimize the work-in-progress across micro-services on cloud and edges. Since work-in-progress is proportional to average delay of each microservice as well as processing delay of workflows, such the objective also corresponds to minimizing the average delay across all workflow types. Specifically, at the end of $k$-th time window, we would like to solve an optimization problem to find the optimal number of consumers for micro-services in the next time window (i.e., $\mathbf{m}(k+1)$) that minimizes the aggregated work-in-progress across all micro-services. The problem is subjected to resource constraints $\mathcal{C}^e$ ($0 \leq e \leq E$) that represents the maximum number of task consumers can be allocated on cloud and on each edge.

$$\begin{aligned}
&\underset{\mathbf{m}(k+1)}{\text{argmin}} && \sum_{e=0}^{E} \sum_{j=1}^{J} w_j^e(k+1) \\
&\text{subject to} && \sum_{j=1}^{J} m_j^e(k+1) \leq \mathcal{C}^e, \forall 0 \leq e \leq E
\end{aligned} \tag{3}$$

For simplicity, if we assume that the objective function of (3) is a linear function of $\mathbf{m}(k+1)$, the optimization (3) is an integer linear programming problem, which is NP-hard.

In fact, as we often see in a complex system that consists of a number of micro-services with complex dependency relationships to support various types of workflows, the formulation of performance metrics (i.e., $\mathbf{w}(k+1)$) by resource configuration (i.e., $\mathbf{m}(k+1)$) is often a non-linear and complex function. As a result, problem (3) could not be solved efficiently by well-known linear programming techniques.

In this paper, we leverage the learned microservice performance model that captures the relationship between $\mathbf{w}(k+1)$ and $\mathbf{m}(k)$ (i.e., the performance model provides a function $\mathbf{w}(k+1) = \boldsymbol{f}(\mathbf{w}(k), \mathbf{m}(k))$) and propose a greedy strategy (Algorithm 1) to efficiently solve the optimization problem (3). Specifically, given the current microservice allocation at time $k$ (i.e., $\mathbf{m}(k)$), for each available task consumer that can be allocated (i.e., the while loop from Line 5-10), the algorithm greedily finds the microservice with the most *benefit* if it is allocated one additional consumer (Line 6). The benefit is defined to be the decrease in the work-in-progress of a microservice, i.e., $w^e_j(k) - \overline{w^e_j(k+1)}$, in which $\overline{w^e_j(k+1)}$ is the predicted work-in-progress of task $j$ microservice on edge $e$ if we allocate one additional consumer to it (i.e., $m^e_j(k+1) = m^e_j(k) + 1$).

**Algorithm 1** Micro-service Work-in-progress Optimization

```
1: procedure μServiceWIPOpt($\mathbf{m}(k)$)
2:     Initialize $\mathbf{m}(k+1) = \mathbf{m}(k)$
3:     $\mathtt{C}_j = \{1, .., J\}$
4:     $\mathtt{C}_e = \{0, .., E\}$
5:     while $\sum_{j=1}^{J} m^e_j(k+1) \leq \mathcal{C}^e (\forall 0 \leq e \leq E)$ do
6:         Find $(j^*, e^*) = argmax_{j \in \mathtt{C}_j, e \in \mathtt{C}_e} [w^e_j(k) - \overline{w^e_j(k+1)}]$
7:         $m^{e^*}_{j^*}(k+1) = m^{e^*}_{j^*}(k+1) + 1$
8:         if $\sum_{j=1}^{J} m^{e^*}_j(k+1) = \mathcal{C}^{e^*}$ then
9:             $\mathtt{C}_e = \mathtt{C}_e \setminus e^*$
10:    Return $\mathbf{m}(k+1)$
```

Please note that at the beginning of Algorithm 1, for simplicity, we initialize $\mathbf{m}(k+1)$ to be equal $\mathbf{m}(k)$. However, in case the current number of consumers $\mathbf{m}(k)$ already reaches the total capacity of cloud and edges (i.e., $\sum_{j=1}^{J} m^e_j(k+1) = \mathcal{C}^e(\forall 0 \leq e \leq E)$), we can initialize $\mathbf{m}(k+1)$ by subtracting a number of consumers from each microservice in the current allocation $\mathbf{m}(k)$ to leave room for re-allocation of consumers: $m^e_j(k+1) = m^e_j(k) - \eta$ if $m_j(k) > \eta$ (in our evaluation, we use $\eta = 3$).

### C. Micro-service Computation Placement

As described in Section III-B, the microservice execution model enables flexible placement of computation to edge- and cloud-based micro-services. In this section, we present our microservice placement strategy based on the system performance model shown in Section V-A.

Our strategy is motivated from the following *invariant* of placing computation across cloud and edge: For task micro-services in a workflow (e.g., `Wf1` in Figure I), once a microservice (e.g., `A`) is placed on the cloud, all of its subsequent micro-services in the workflow (i.e., `B` and `C`) are also placed on the cloud (i.e., to avoid unnecessary and costly round-trip communications between cloud and edge).

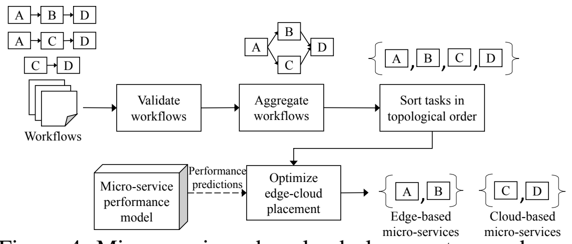

A
B
D
A
C
D
C
D
Workflows
Validate workflows
A
B
C
D
Aggregate workflows
A, B, C, D
Sort tasks in topological order
Micro-service performance model
Performance predictions
Optimize edge-cloud placement
A, B
Edge-based micro-services
C, D
Cloud-based micro-services

Figure 4: Micro-service edge-cloud placement procedure.

The microservice placement procedure for each branch of an edge and cloud is presented in Figure 4. First, all the workflow types that correspond to data uploaded from the edge are validated to ensure that they are in DAG format. Then, all workflow types are aggregated into a single DAG graph. After that, all the tasks in the aggregated graph are sorted in topological order. The next step is to find an *edge-cloud cut* to partition the set of tasks into two sets: one set whose computation is placed on the edge-based micro-services and another set whose computation is placed on the cloud-based micro-services. The placement procedure iterates over the tasks in the topological order obtained from previous step. At the $\iota$-th iteration ($1 \leq \iota \leq J$), the $\iota$-th task in the topological order is considered as the edge-cloud cut. It means that the computation of all tasks up to $(\iota - 1)$-th in topological order is placed on the edge-based micro-services, and the computation of those from $\iota$-th is placed on the cloud-based microservices.

To evaluate the placement of a task as the edge-cloud cut, we consider whether such placement helps to: *(i)* minimize the average delays of micro-services that involve in the placement (i.e., *min delay* criteria), and *(ii)* minimize the communication cost between edge and cloud (i.e., *min communicaiton* criteria). We leverage the microservice performance model learned from Section V-A to quantify these two criteria. In particular, at the current time $k$, the performance model is used to make predictions about work-in-progress of each microservice in the next time window $\mathbf{w}(k+1)$. By the Little Law, $\mathbf{w}(k+1)$ can be used to estimate the processing delay by summing up the work-in-progress requests of all task micro-services (i.e., quantifying min delay). As we consider to use task $\iota$-th in the topological order as the edge-cloud cut between an edge $e$ ($1 \leq e \leq E$) and the cloud, the communication cost between edge- and cloud-based microservices can be estimated by summing up the number of work-in-progress of all microservices whose computation would be placed at the edge $e$ (i.e. the microservices up to $(\iota-1)$-th in topological order): $\sum_{j=1}^{\iota-1} w^e_j(k+1)$ (or 0 if $\iota = 1$, or computation of all microservices are placed on the cloud). That is because these work-in-progress requests will be processed by microservices at the edge and

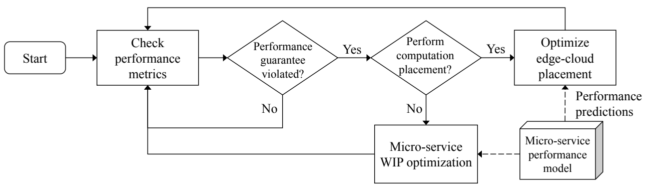

Start
Check performance metrics
Performance guarantee violated?
Yes
No
Perform computation placement?
Yes
No
Optimize edge-cloud placement
Performance predictions
Micro-service performance model
Micro-service WIP optimization

Figure 5: BRACELET's joint resource allocation procedure.

the intermediate results will be transmitted to the cloud for processing by subsequent microservices (i.e., from the $\iota$-th microservice in topological order).

In order to balance between the benefit of reducing processing delay and the communication cost, we use the following function to measure the "goodness" of using task $\iota$-th in the topological order as the edge-cloud cut between an edge $e$ and the cloud:

$$\rho_{(\iota,e)}(k+1) = \alpha \cdot \sum_{j=1}^{\iota-1} w_j^e(k+1) + \beta \cdot \sum_{j=\iota}^{J} w_j^0(k+1) \quad (4)$$

In Equation 4, while the first component captures potential communication cost, both components help to capture potential processing delay of the placement. $\alpha$ and $\beta$ are normalization factors used to balance between two components. As we iterate through the tasks in the topological order, we report the task whose placement as the edge-cloud cut helps minimize the above goodness function and use it as the edge-cloud cut in placement decision.

### D. BRACELET*'s Joint Resource Allocation Procedure*

While both microservice placement and WIP optimization procedures rely on the performance model and both can be used to control system performance, placement procedure is more efficient than WIP optimization. In particular, the first three steps of the placement procedure can be done offline and the online step (i.e., optimize edge-cloud placement) only needs to iterate over all types of tasks in an edge-cloud branch. On the other hand, WIP optimization needs to iterate over all task micro-services (edge- and cloud-based ones) *for each* available consumer.

Thus, we combine the two procedures into a joint resource allocation procedure (Figure 5). Once started, the procedure keeps running as long as the system operates and periodically checks system performance metrics (via monitoring layer) to see if certain performance guarantee is violated. If there is violation, the procedure will first try to invoke the more efficient edge-cloud placement procedure to mitigate the violation. The procedure will keep trying with edge-cloud placement optimization for a predefined number of times before invoking the more expensive work-in-progress optimization, in case the performance violation could not be mitigated.

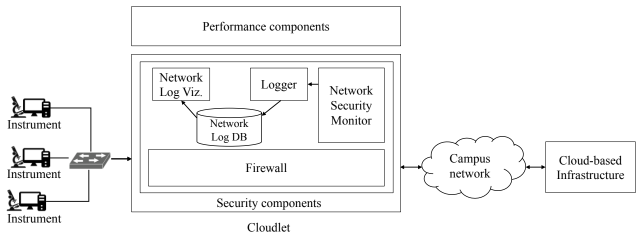


Figure 6: BRACELET' security component design.

## VI. SYSTEM VALIDATION

### A. System Implementation

We implement BRACELET by extending the implementation of the existing cloud-based micro-service infrastructure [3, 4] to the edges. The whole edge-cloud infrastructure cluster is managed by a single Kubernetes [7] container orchestration engine. Specifically, the cloud-based system is deployed on a cluster of two nodes, each node is equipped with an Intel Xeon quad core processor, 1.2Ghz per core, and 16GB of RAM. Two cloudlets, each cloudlet is equipped with Intel Core i7 CPU 3.4Ghz and 8GB of RAM, are connected to the cloud-based system as remote nodes in the Kubernetes cluster. Each cloudlet has its own locality tagging to differentiate it from other cloudlet and cloud-based nodes. This locality tag is used by cloud controller to place computation specifically to the microservices running on the edge. The main reasons why we deploy edge- and cloud-based components in a single Kubernetes cluster are two fold: i) to simplify service discovery between microservice running on edges and cloud (i.e., microservices run on the same overlay network managed by the Kubernetes cluster), and ii) to provide a single, global view of resources for the cloud controller.

Each microservice is implemented with a RabbitMQ [8] request queue and a set of Docker container-based task consumers that are deployed as a ReplicationController[6] set on Kubernetes. TDS service is based on Apache Zookeeper [9] coordination system to ensure strong consistency and high availability. We configure Zookeeper and RabbitMQ using ensemble and cluster mode respectively so that we have a Zookeeper and RabbitMQ endpoint on each edge and cloud side (i.e., to improve availability and enable seamless communication between microservices). Monitoring layer's implementation is similar to the one in [4], and we use Tensorflow [10] to build microservice performance model used in the adaptation layer.

### B. BRACELET' *Security Design and Implementation*

BRACELET' security components (Figure 6) are designed to help protect vulnerable scientific instruments once they are connected to the edge-cloud cyberinfrastructure. They consist of a software firewall that is configured with *whitelisting rules* to enable only data traffic from instruments to the cloud and certain control traffic from the cloud to the cloudlet. Furthermore, each cloudlet also includes a network security monitor component to listen to and capture meta-data of all network traffic in and out of the cloudlet. The security monitor component is also capable of applying customizable scripts to filter and analyze network traffic to detect and alert of potential attacks. All network monitoring logs are collected, parsed, and transformed by a *logger* component, and stored into a network logs database. Real-time network traffics and statistics can be queried and visualized to BRACELET's admin by the *network log visualization* component. In our implementation, we use Bro [11] as the network security monitor at cloudlet and use ELK stack [12] (i.e., Elastic-Logstash-Kibana) for logging, storing, and visualizing the collected network security logs.

In addition to data driven security monitoring and detection at cloudlet, all vulnerable scientific instruments are connected to private instrument network via a managed switch so that instrument's MAC layer address is checked to ensure that the instrument can only talk to cloudlet and not to other peer instruments. At application level, users are required to login on each instrument in order to upload data, and the login sessions are additionally verified with *instrument reservation database* as part of the two-factor authentication process.

## VII. EVALUATION

### A. *Evaluation Settings*

In terms of the workload, we use the MDP workflow ensemble [3, 4] that supports processing experimental data generated by digital microscopes. MDP consists of three types of workflows (numbered 1, 2, and 3) and four types of tasks (named `A`, `B`, `C`, and `D`).

In order to obtain training data to train the performance model, we let the system runs through a *bootstrapping process* in which we randomly vary the arrival rates of incoming requests of different workflow types and randomly vary the allocation of consumers across microservices (i.e., $\mathbf{m}(k)$) as well as the computation placement of tasks in a workflow across cloud and edges. We record the actual performance output $\hat{\mathbf{w}}(k+1) = \{\hat{w}_j^e(k+1)\}$ of microservices to use as the ground-truth. As a result, our collected dataset consists of tuples $\mathcal{D} = \{(\mathbf{w}(k), \mathbf{m}(k), \hat{\mathbf{w}}(k+1))\}$, in which

[6]ReplicationController is a concept in Kubernetes that consists of a set of replicas of a container. Kubernetes helps to ensure that a specified number of these container replicas (i.e., corresponds to $m_j(k)$ in our case) are running at any time.

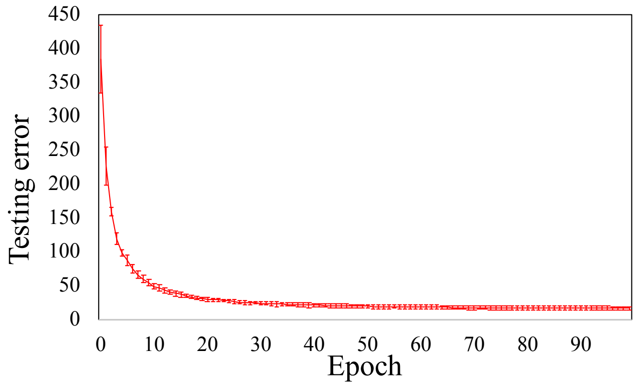


Figure 7: Testing error of microservice performance model over training epochs.

$\mathbf{w}(k), \mathbf{m}(k)$ are input and $\hat{\mathbf{w}}(k+1)$ is true output. We use this data to train a performance model of microservices $\boldsymbol{f}$ to predict work-in-progress of microservices in the next time window, given the current work-in-progress and allocation of consumers: $\mathbf{w}(k+1) = \boldsymbol{f}(\mathbf{w}(k), \mathbf{m}(k))$.

We design the neural network model with two hidden layers and an output layer (i.e., total number of layers $n = 3$), with the number of hidden neurals $S^1$ and $S^2$ both equal 128, and $\boldsymbol{f}^1$ and $\boldsymbol{f}^2$ are ReLU function. For training, we set learning rate as 0.001, batch size as 100, and use 100 training epochs. We split the collected dataset $\mathcal{D}$ into two parts: 80% for training and 20% for testing.

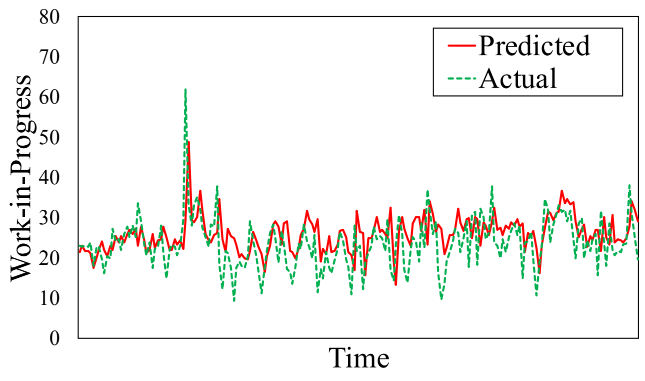
To evaluate BRACELET's microservice placement and work-in-progress optimization strategies, we emulate up to 5x spikes of workflow requests to BRACELET (both cloud and edge sides) and measure how effective our proposed approaches is, compared to related approaches. BRACELET's resource allocation procedure is invoked if average delay $\bar{\mathbf{d}}(k)$ exceeds performance guarantee of 20 seconds.

### B. *Evaluation of Microservice Performance Model*

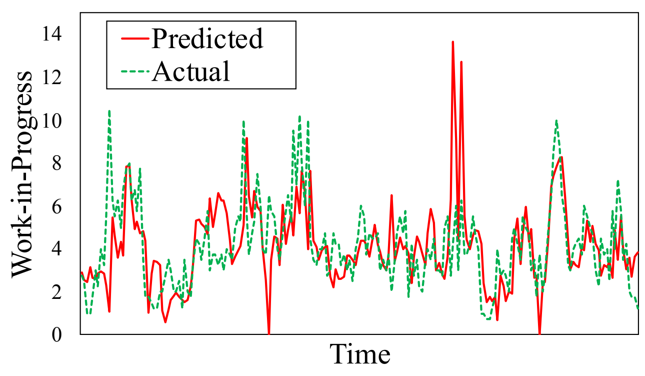
Figure 7 shows the testing error of the performance model over training epochs. It shows that the neural network model quickly converges to a stable testing performance. The result is also consistent (i.e., demonstrated through decreasing variance over time) as we repeat the test multiple times, each time with different initializations of neural network model. This result helps to verify the effectiveness of using neural network to capture performance of microservices.

While the result in Figure 7 shows the effectiveness of performance model *quantitatively*, to see the effectiveness of the model in action *qualitatively*, we use the trained model to make predictions on future performance of microservices $\mathbf{w}(k+1)$ and see how close the predictions are compared to the actual performance $\hat{\mathbf{w}}(k+1)$.

The results in Figure 8 show that the performance predictions by the trained model can be used effectively to capture the *average work-in-progress* of microservices on cloud (Figure 8a) and on edge (Figure 8b). This is important

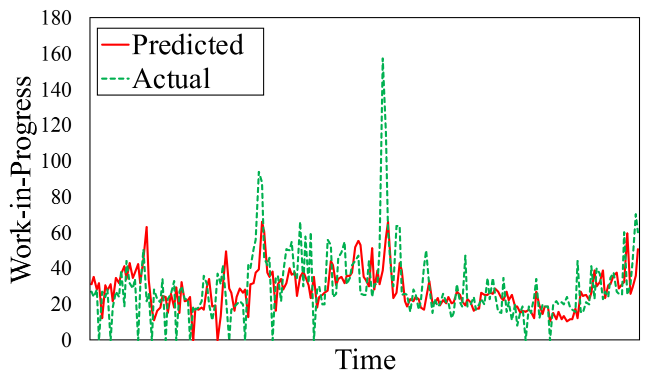


(a) Predictions of cloud-based average work-in-progress.

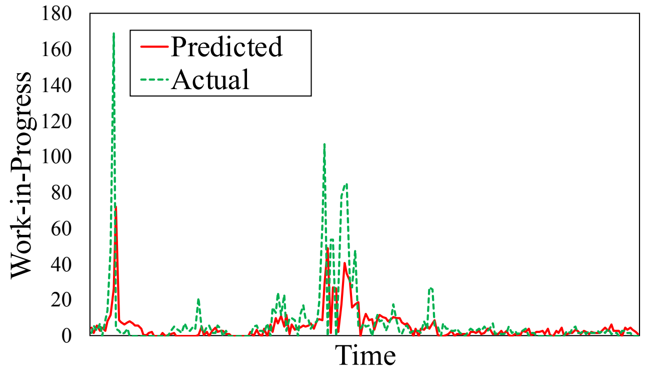


(b) Predictions of average work-in-progress on cloudlet `E2`

Figure 8: Effectiveness of microservice performance model on capturing average work-in-progress of microservices on cloud and edges.



(a) Predictions of work-in-progress of microservice `D` on the cloud.



(b) Predictions of work-in-progress of microservice `C` on cloudlet `E1`

Figure 9: Effectiveness of microservice performance model on capturing work-in-progress of individual microservices on cloud and edges.

since, according to Little's law, the aggregated work-in-progress is proportional to the end-to-end processing delays of workflows. And as we try to find an optimal allocation of consumers over microservices to minimize aggregated work-in-progress (i.e., optimization problem (3)), we would expect our objective function is closely related to the actual aggregated work-in-progress, and ultimately, to the average processing delays of workflows.

In addition to evaluating performance model on aggregated performance, we also want to see how the model performs when predicting work-in-progress of *individual* microservices. The results on Figure 9 show that the model is able to predict accurately the near future performance (measured by work-in-progress) of individual microservices, including both cloud-based (Figure 9a) and edge-based (Figure 9b) microservices. This result on predicting individual performance is important, since the predicted work-in-progress of individual microservices are used in both resource allocation (i.e., Algorithm 1) and computation placement (i.e., Section V-C). Especially, as shown in Figure 9, the model can predict very effectively and *timely* the spikes in work-in-progress of microservices. This is vital in making accurate and timely resource allocation and placement decisions when dealing with dynamic and bursty workload situations.

## C. Evaluation of Microservice Computation Placement

For this evaluation, we fix the number of consumers of micro-services $\mathbf{m}(k)$ and evaluate how BRACELET's microservice computation placement strategy can handle sudden spike in the incoming requests. We compare our placement strategy with other related approaches:

- *Bandwidth-optimized*: Initially, all requests are handled by cloud-based micro-services. When performance guarantee is violated, the processing of requests that arrive from an edge is offloaded to the edge-based micro-services. As shown in [6], this approach helps to improve the efficiency of cloud resource utilization when serving the peak workload.
- *Delay-optimized*: This strategy is often employed in mobile cloud computing scenario to optimize delays of processing requests coming from edges. Initially, requests arriving from an edge are handled by edge-based micro-services. When performance violation occurs, the processing of requests is offloaded to the cloud-based micro-services.

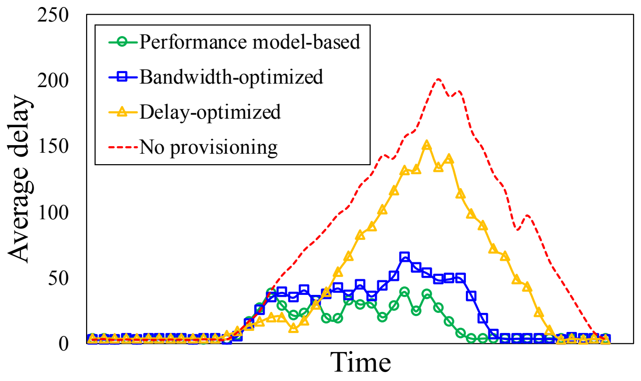


Figure 10: Effectiveness of BRACELET's microservice placement strategy compared with others.

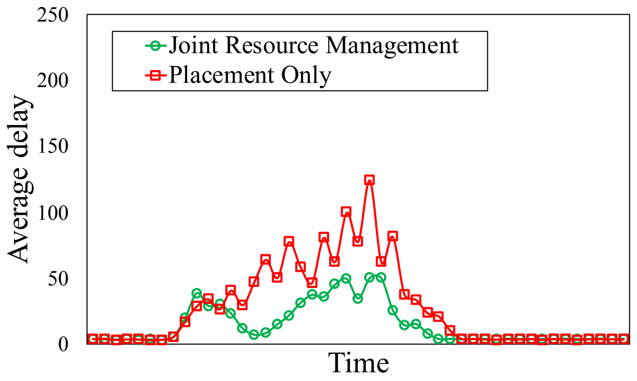


Figure 11: Effectiveness of BRACELET's joint resource allocation procedure.

The result in Figure 10 shows that our proposed computation placement strategy outperforms other approaches by dynamically evaluating different placement options using the accurate performance model. The dynamism of placement decisions is captured by the small ups and downs in average delay when using our strategy (i.e., green-circle line). Delay-optimized scheme performs poorly since it creates congestion on cloud-based micro-services when performance violation occurs.

### D. *Evaluation of Joint Resource Allocation Procedure*

It is intuitive that, given additional consumers to allocate, the microservice WIP optimization procedure can help improve the result when using only microservice computation placement. However, in private, on-premise cloud infrastructure, the total number of consumers can be bounded by the resource capacity. In this evaluation, we show that, even *without* any additional consumer to allocate (i.e., the total number of consumers reaches the limit), the joint procedure introduced in Section V can still achieve better result by smartly re-allocating consumers among microservices to the ones that are most in need, compared to when using only microservice computation placement.

In particular, given an initial allocation of consumers over micro-services, when performance guarantee is violated due to a sudden spike in the workload, the joint procedure will first try with changing computation placement. After a number of attempts (i.e., 3 retries in our evaluation), without any additional consumers, the joint procedure will try to re-allocate the current set of consumers to micro-services that are most beneficial from such re-allocation using Algorithm 1. Results in Figure 11 show that such the joint approach greatly helps to reduce the affect of the spike in workload, compared to when using only microservice computation placement, even in the extreme case when there is no additional resource.

## VIII. Related Work

Related work on *cyber-infrastructure* [3, 13, 14] has mainly focused on cloud-based, two-tier architecture and lacks of support for vulnerable scientific instruments running on out-of-date operating systems. In this paper, we design *the first* microservice based edge-cloud architecture for cyberinfrastructure that seamlessly extends cloud-based infrastructure to the edges to help connect and protect otherwise disconnected and vulnerable instruments.

In terms of *computation offloading* in the cloud-edge architecture, there have been a number of related work in mobile computing domain [15] that aim to minimize execution time or preserve energy on the mobile endpoints. Tong et al. [6] propose a workload placement algorithm to decide which edge cloud servers mobile programs are placed on (i.e., placement) and how much computational capacity needed (i.e., scaling). Tan et al. [16] propose a general model for deciding when and where to offload a job from a mobile user. Wang et al. [17] investigate the assignment and the scheduling of tasks over multiple cloudlets. Most of related work, however, only deals with workloads of independent tasks (and in a lot of case, with known task profiles). In this paper, we are dealing with dependent tasks from multiple workflow types and we propose a novel resource placement and scaling across edge cloud infrastructure using a microservice performance model that does not require any profiling of tasks.

In terms of *resource scaling of cloud application*, there is also related work [4, 18, 19, 20] on using predictive model (especially using machine learning techniques) to accurately predict performance and resource demand of applications to make informed decisions on resource scaling and reconfiguration. The main difference between those approaches and our proposed approach is that we model the system performance at the granularity of *individual microservices* (using work-in-progress and scaling of microservices), instead of traditional performance metrics such as delay, utilization, and resource models such as CPU, RAM. In addition, we leverage our performance model to support ***both*** resource scaling and computation placement decisions.

## IX. Conclusions and Future Work

In conclusions, we have presented *the first* edge-cloud microservice cyber-infrastructure that tackles both performance and security challenges of scientific instruments' lifetime connectivity. We also present a robust resource allocation mechanism that is based on a microservice performance model that tackles both computation placement and resource scaling of microservices. For future work, we plan to study the edge-cloud architecture with more layers in the hierarchy and research on methods to reduce training effort to train the performance model.

## Acknowledgment

This research was funded by the National Science Foundation NSF, award number 1443013 and 1659293. The opinions, findings and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the view of the National Science Foundation.

## References

[1] A. R. Ferguson, J. L. Nielson, M. H. Cragin, A. E. Bandrowski, and M. E. Martone, "Big data from small data: data-sharing in the'long tail'of neuroscience," *Nature neuroscience*, vol. 17, no. 11, p. 1442, 2014.

[2] J. Holdren, "Materials genome initiative for global competitiveness," *National Science and Technology Council OSTP*, 2011.

[3] P. Nguyen *et al.*, "4ceed: Real-time data acquisition and analysis framework for material-related cyber-physical environments," in *Cluster, Cloud and Grid Computing (CCGRID), 2017 17th IEEE/ACM International Symposium on*, 2017, pp. 11–20.

[4] P. Nguyen and K. Nahrstedt, "Monad: Self-adaptive microservice infrastructure for heterogeneous scientific workflows," in *Autonomic Computing (ICAC), 2017 IEEE International Conference on*, 2017, pp. 187–196.

[5] "User study and survey on material-related experiments," http://hdl.handle.net/2142/94738, 2016, [Online; accessed April 30, 2018].

[6] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*, 2016, pp. 1–9.

[7] "Kubernetes," https://kubernetes.io/, accessed: 2018-05-10.

[8] "RabbitMQ," https://www.rabbitmq.com/, accessed: 2018-05-10.

[9] "Apache Zookeeper," https://zookeeper.apache.org/, accessed: 2018-05-10.

[10] "Tensorflow," https://www.tensorflow.org/, accessed: 2018-05-10.

[11] "Bro Network Security Monitor," https://www.bro.org/, accessed: 2018-05-10.

[12] "ELK Stack," https://www.elastic.co/elk-stack, accessed: 2018-05-10.

[13] M. McLennan and R. Kennell, "Hubzero: a platform for dissemination and collaboration in computational science and engineering," *Computing in Science & Engineering*, vol. 12, no. 2, 2010.

[14] M. S. Mayernik, G. S. Choudhury, T. DiLauro, E. Metsger, B. Pralle, M. Rippin, and R. Duerr, "The data conservancy instance: Infrastructure and organizational services for research data curation," *D-Lib Magazine*, vol. 18, no. 9/10, 2012.

[15] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[16] H. Tan, Z. Han, X.-Y. Li, and F. C. Lau, "Online job dispatching and scheduling in edge-clouds," in *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*, 2017, pp. 1–9.

[17] L. Wang, L. Jiao, D. Kliazovich, and P. Bouvry, "Reconciling task assignment and scheduling in mobile edge clouds," in *Network Protocols (ICNP), 2016 IEEE 24th International Conference on*, 2016, pp. 1–6.

[18] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "Agile: Elastic distributed resource scaling for infrastructure-as-a-service." in *Autonomic Computing (ICAC), 2013 USENIX International Conference on*, vol. 13, 2013, pp. 69–82.

[19] A. Matsunaga and J. A. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010, pp. 495–504.

[20] R. C.-L. Chiang, J. Hwang, H. H. Huang, and T. Wood, "Matrix: Achieving predictable virtual machine performance in the clouds." in *Autonomic Computing (ICAC), 2014 USENIX International Conference on*, 2014, pp. 45–56.