Task Partitioning and Orchestration on Heterogeneous Edge Platforms: The Case of Vision Applications

Dapeng Lan, Amir Taherkordi, Frank Eliassen, Lei Liu, Stéphane Delbruel, Schahram Dustdar, Yang Yang

▶ **To cite this version:**

Dapeng Lan, Amir Taherkordi, Frank Eliassen, Lei Liu, Stéphane Delbruel, et al.. Task Partitioning and Orchestration on Heterogeneous Edge Platforms: The Case of Vision Applications. IEEE Internet of Things Journal, 2022, 9 (10), pp.7418-7432. 10.1109/JIOT.2022.3153970 . hal-04901292

HAL Id: hal-04901292
https://hal.science/hal-04901292v1

Submitted on 24 Jan 2025

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial - ShareAlike 4.0 International License

Task Partitioning and Orchestration on Heterogeneous Edge Platforms: The Case of Vision Applications

Dapeng Lan, *Member, IEEE*, Amir Taherkordi, *Member, IEEE*, Frank Eliassen, Lei Liu, *Member, IEEE*, Stéphane Delbruel, Schahram Dustdar, *Fellow, IEEE*, and Yang Yang, *Fellow, IEEE*

***Abstract*—Running computer vision applications, such as 3-D simultaneous localization and mapping (SLAM), on mobile devices requires low-latency responses and a massive amount of computation. Edge computing has been introduced to move Cloud features closer to end users, providing necessary computing and network resources for end devices. The heterogeneous edge devices, with different hardware architectures (e.g., CPUs and GPUs) and runtime environments, provide diverse resources to support processing tasks from end devices, resulting in different costs and quality of services. How to partition these computing tasks and distribute them over these heterogeneous hardware nodes is still an open research question. Considering these inherently heterogeneous hardware architectures, new approaches for service orchestration and task scheduling are required to meet the service-level agreement and reduce the overall cost of the system (e.g., facility utilization cost). This article presents a system framework, EDGEVISION, for computer vision applications partitioning and orchestration on heterogeneous edge computing platforms considering both CPUs and GPUs. EDGEVISION abstracts the heterogeneous hardware resources and the task runtime environments and divides the application into separate tasks to be orchestrated and deployed into the heterogeneous edge nodes. We also propose two scheduling algorithms in our framework, minimum latency task scheduling and minimum cost task scheduling, aiming to minimize the processing latency and the overall system cost. We evaluate our framework by implementing the edge-based 3-D SLAM application in our real testbed with ten heterogeneous edge devices. Evaluations show that EDGEVISION can efficiently minimize the processing latency and the system overall cost and achieve up to 30% decrease in task processing latency and 15% more cost saving compared to the State-of-the-Art baselines.**

***Index Terms*—3-D simultaneous localization and mapping (SLAM), application partitioning, computer vision, heterogeneous edge computing, orchestration.**

Manuscript received July 27, 2021; revised December 6, 2021 and January 18, 2022; accepted February 6, 2022. Date of publication February 25, 2022; date of current version May 9, 2022. This work was supported in part by the Norwegian Research Council through the DILUTE Project under Grant 262854/F20; in part by the National Key Research and Development Program of China under Grant 2020YFB2104300; in part by the Major Key Project of Peng Cheng Laboratory under Grant PCL2021A15; in part by the Joint Funds of the National Natural Science Foundation of China under Grant U21B2002; in part by the National Natural Science Foundation of China under Grant 62001357; in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2020A1515110079; in part by the China Postdoctoral Science Foundation under Grant 2021M692501; in part by the Fundamental Research Funds for the Central Universities under Grant XJS210107; and in part by the Key Projects of Science and Technology of Henan Province under Grant 222102210043. *(Corresponding author: Lei Liu.)*

Dapeng Lan, Amir Taherkordi, and Frank Eliassen are with the Department of Informatics, University of Oslo, 0316 Oslo, Norway (e-mail: dapengl@ifi.uio.no; amirhost@ifi.uio.no; frank@ifi.uio.no).

Lei Liu is with the State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an 710071, China, and also with the Research Center of Trusted Digital Economy, Xidian Guangzhou Institute of Technology, Guangzhou 510555, China (e-mail: tianjiaoliulei@163.com).

Stéphane Delbruel is with LaBRI, University of Bordeaux, 33405 Talence, France (e-mail: stephane.delbruel@labri.fr).

Schahram Dustdar is with the Distributed Systems Group, Technische Universitat Wien, 1040 Vienna, Austria (e-mail: dustdar@dsg.tuwien.ac.at).

Yang Yang is with the Shanghai Institute of Fog Computing Technology, ShanghaiTech University, Shanghai 201210, China, also with the Department of Broadband Communication, Peng Cheng Laboratory, Shenzhen 518055, China, and also with Shenzhen SmartCity Technology Development Group Company Ltd., Shenzhen 518046, China (e-mail: yangyang@shanghaitech.edu.cn).

Digital Object Identifier 10.1109/JIOT.2022.3153970

I. INTRODUCTION

IN THE last decade, Cloud computing has been serving well for the Internet of Things (IoT) and data processing applications with abundant computing and storage resources, on-demand self-service, and broad network access [1]. With the coming era of 5G, emerging mobile applications with video processing tasks, such as VR/AR, autonomous driving, and 3-D simultaneous localization and mapping (SLAM), demand more computing resources with low-latency processing requirements [2], [3]. Nowadays Cloud-centric architectures that host most computing jobs in data centers, cannot satisfy such requirements. Edge (also known as fog) computing has been a new paradigm shifting the Cloud characteristics to the network edge, closer to the field of applications [4]–[6]. Although both edge computing and fog computing move computing and storage to the edge of the network and closer to end devices, these paradigms are not the same [7]. Fog seeks to realize seamless and continuous computing services from the cloud to end devices, while edge computing tends to be limited to computing at the edge [8]. We focus on edge computing in this article. In this way, the end devices can offload computing tasks to the nearby edge devices and receive fast response results without edge-to-core network communication delays [9].

Computation offloading in edge computing requires partitioning an application into different tasks and scheduling these tasks over the edge devices, which poses many research challenges [10]. Application partitioning means to separate the components of the mobile applications in the distributed

2327-4662 © 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on May 09,2022 at 08:13:51 UTC from IEEE Xplore. Restrictions apply.

environments [11]. Liu *et al.* [12] have provided a taxonomy of application partitioning algorithms in mobile cloud computing, including portioning granularity (task, component, method, etc.), objectives (improving performance, saving cost, etc.), partitioning models (graph, liner, etc.), allocation decisions (offline, online, etc.), and so on. Some works [13]–[15] have studied partitioning the application between the mobile devices and the Cloud, e.g., MAUI [13] offloads some parts of the application execution from the phone to the Cloud. A few works [16], [17] considered application partitioning in mobile-edge computing, e.g., Wu *et al.* [16] proposed a dynamic algorithm to find the optimal partitioning plan between the edge and the Cloud. However, these State-of-the-Art (SotA) works only consider homogeneous computing resources and fail to consider the heterogeneous characteristics of hardware architectures (e.g., CPUs and GPUs). In this article, we focus on the software component level of application partitioning, but our methods can also be extended to the method and module levels.

Heterogeneous edge computing platforms face many challenges concerning orchestration frameworks, requiring more dynamic task schedulers than for cloud platforms. For example, one challenge is the heterogeneity of edge devices that have various hardware architectures and runtime environments. However, today's frameworks or architectures do not consider the heterogeneous characteristics of hardware architectures (such as CPUs and GPUs) and the service-level agreement (SLA) with minimum cost, when deploying and orchestrating tasks. Deploying tasks into different hardware architectures will result in different costs and latency. For instance, using GPUs to calculate artificial intelligence (AI) tasks will be 10 to 100 times faster than only using CPUs. Besides, orchestrating the tasks without considering the resource metrics (such as computing resources and network resources like bandwidth) will result in suboptimal resource allocations and suboptimal SLAs.

The above challenges motivate us to devise a novel system framework considering the heterogeneous aspects of hardware architectures and introduce a dynamic orchestration of computing and networking resources to guarantee the Quality of Service (QoS) and reduce the system overall cost. We summarize our contributions as follows.

1) We have designed and implemented a vision application partitioning and orchestration framework, called EDGEVISION, for heterogeneous edge computing. EDGEVISION abstracts the heterogeneous hardware resources and the runtime environments of tasks, considering both CPU and GPU computing platforms.
2) We have devised a dynamic task topology generation scheme and two scheduling algorithms: task scheduling with minimum latency and minimum cost, which, when combined, can proactively minimize the task processing latency and the system overall cost in heterogeneous edge computing platforms.
3) We have evaluated EDGEVISION by implementing the edge-based 3-D SLAM application on a real testbed consisting of ten heterogeneous edge devices. The results of the evaluation show that EDGEVISION can efficiently achieve 30% decrease in latency for processing the tasks and 15% decrease of the system overall cost, compared to the SotA baselines.

The remainder of this article is organized as follows. Section II discusses the related work on application partitioning in edge computing, orchestration for edge computing, and distributed systems and heterogeneous computing. We present the detailed design of the vision application partitioning and orchestration framework, EDGEVISION, in Section III. In Section IV, we describe the details of the problem formulation. Finally, the performance of our proposed framework is evaluated in Section VI. We conclude this article and discuss the future work in Section VII.

II. RELATED WORK

In this section, we split the related work into two major branches: 1) application partitioning and orchestration in edge computing and 2) task allocation in heterogeneous computing. Table I summarizes the comparison of existing related work.

A. Application Partitioning and Orchestration in Edge Computing

Application portioning has been well studied in the context of mobile cloud computing [12]–[15]. Some works [16], [17] have adapted the application partitioning techniques from mobile cloud computing to mobile-edge computing that has more heterogeneous resources and dynamic environment changes. Wu *et al.* [16] designed a dynamic algorithm to find the optimal partitioning plan between the edge and the cloud while reducing the total cost to the most possible degree. Cao *et al.* [17] presented a partitioning model that parallelizes the computations and fully utilizes the computational resources at the edge and end devices, in the context of future 5G-based edge computing. Some other works proposed totally new edge computing frameworks that partly deal with application partitioning. For instance, Cheng *et al.* introduced the FogFlow framework in [18] for cloud and edge platforms, dividing the application into edge and cloud by extending the dataflow programming model and utilizing the *NGSI* standard. Han *et al.* [19] introduced a learning-based scheduling framework for Kubernetes-oriented edge-cloud systems that leverages the multiagent actor–critic algorithm and deduced the orchestration dimensionality by stepwise scheduling. However, these works are built upon homogeneous hardware architectures and failed to consider the heterogeneous characteristics of hardware architectures (e.g., CPUs and GPUs) to provide a practical application partitioning and service orchestration solution.

B. Task Allocation and Heterogeneous Computing

Dealing with heterogeneity among various computing nodes has been identified as a critical challenge in distributed computing systems. Some solutions have been proposed in the past targeting either network heterogeneity or resource heterogeneity. For instance, Yang *et al.* [20] introduced a multitask and multihelper framework in heterogeneous fog networks and utilized a game theory method called paired offloading of multiple tasks to solve the optimization problem. However, the

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on May 09,2022 at 08:13:51 UTC from IEEE Xplore. Restrictions apply.

TABLE I
COMPARISON OF EXISTING RELATED WORK

Reference	Partitioning & Orchestration	Heterogeneity	Key Metrics	Evaluation Methods	Main Contribution
[13]	Partitioning .NET Applications between phone and the Cloud	Two devices with only CPU	Energy saving	Implementation	MAUI, a system that enables fine-grained energy-aware offload of mobile code to the infrastructure.
[16]	Task partitioning	Local and remote devices with only CPU	Execution time and energy consumption	Simulation	MCOP dynamically partitions a given application effectively into local and remote parts while reducing the total cost.
[17]	Task partitioning	1000 mobile devices and 100 servers with only CPU	Average completion time	Simulation	This work presents a partitioning model that parallelizes the computations and fully utilizes the computational resources in the edge and end devices, in the context of future 5G-based edge computing.
[18]	Task partitioning based on Container	Resources are vertically divided as cloud, edge nodes, and devices, with only CPU	Response time and throughput	Implementation	FogFlow framework for cloud and edge platforms, dividing the application into edge and cloud by extending the dataflow programming model and utilizing the *NGSI* standard.
[19]	Task partitioning based on Container	Homogeneous edge and cloud cluster with only CPU	Throughput rate and reduce cost	Implementation	This paper introduces KaiS, a learning-based scheduling framework for edge-cloud systems to improve the long-term throughput rate of request processing.
[20]	Task partitioning	Heterogeneous network and computing resources with only CPU	Average delay and delay reduction ratio	Simulation	This work introduces a multi-task and multi-helper framework in heterogeneous fog networks and utilizes a game theory-based method called paired offloading of multiple tasks to solve the optimization problem.
[11]	Task partitioning	Heterogeneous network and computing resources with both CPUs and GPUs	Model training time	Implementation	BytePS can leverage spare CPU and bandwidth resources in the cluster to accelerate distributed DNN training tasks running on GPUs and provide a communication framework that is proved optimal.
[21]	Function partitioning	Heterogeneous network and computing resources with both CPUs and GPUs	Makespan	Implementation	Delta is a system that unifies heterogeneous computing environments while also controlling the flow of various function execution requests.
[22]	Code partitioning	Heterogeneous computing resources with CPU and GPU in HPC	Runtime overhead	Implementation	This paper presents a package for running OpenMP, C++ and unmodified OpenCL applications on clusters with many GPU devices.
[23]	Code partitioning	Heterogeneous computing resources with GPU and FPGA in HPC	Data transfer latency	Implementation	This work proposes an OpenCL-enabled data movement mechanism for accessing the GPU's global memory directly and demonstrates how to use it to build cooperative GPU-FPGA computing.
Our work	Task partitioning based on Container	Heterogeneous computing resources with CPU and GPU in Edge	Overall cost and latency	Implementation	Our work proposes a system framework, EDGEVISION, for computer vision applications partitioning and orchestration on heterogeneous edge computing platforms considering both CPUs and GPUs.

above work suffers from two major limitations: 1) it assumes that tasks require only homogeneous computing resources which is not true in nowadays' emerging edge computing applications and 2) it assumes the tasks are always compatible with the edge devices which is not necessarily true in edge devices with multiple runtime environments, such as operating systems (OSs) and library dependencies.

We can see that most of the current GPU frameworks are being researched in a high-performance computing context. Some other GPU frameworks focus on different partitioning methods and applications fields. For instance, Barak *et al.* [22] proposed many GPUs package (MGP) which allows parallel OpenMP, C++, and unmodified OpenCL applications to operate transparently on several heterogeneous GPU devices in a cluster. Reference [23] provides an OpenCL-enabled data movement mechanism for accessing the GPU's global memory directly and demonstrates how to use it to build cooperative GPU-FPGA computing. However, these works focus on code level and assume the heterogeneous CPUs/GPUs are deployed in a data center rather than in an edge computing environment. A few works consider the heterogeneous computing resources (e.g., GPUs), such as the work by Jiang *et al.* [11], who presented a new distributed deep neural network (DNN) training architecture leveraging sparing CPU, GPU, and bandwidth resources. However, it focuses on only the DNN training field and differs from our work which focuses on application partitioning and orchestration intending to minimize the overall system cost. The most recent and similar work to ours is [21], which presents Delta, a system that unifies heterogeneous computing environments while also controlling the flow of various function execution requests. However, this work is focused on the functional level of the program and on the three

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on May 09,2022 at 08:13:51 UTC from IEEE Xplore. Restrictions apply.

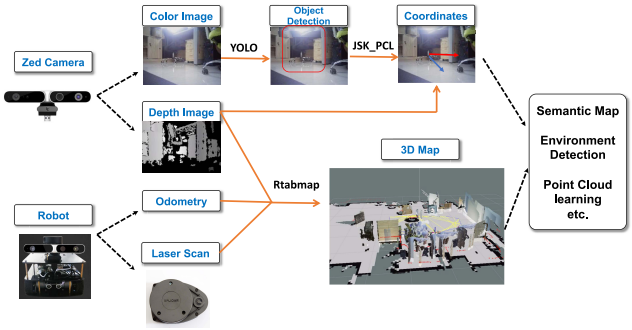

Fig. 1. Our use case 3-D SLAM, a computer vision application running on edge devices.

applications: 1) a parallel counter (ParCount); 2) a matrix multiplication (MatMul); and 3) an I/O heavy application (FileIO), while we focus on tasks running on containers, and the vision application.

To the best of our knowledge, we are the first considering both application partitioning and orchestration on heterogeneous edge computing platforms with CPUs and GPUs, and also evaluating the framework in a real testbed rather than by simulation, which contributes valuable inputs to the related research fields.

III. Detailed EdgeVision Design

In this section, we first introduce our use case, the 3-D SLAM application, which serves as an example for the EdgeVision design. Then, we introduce the details of the EdgeVision framework.

A. Use Case

We use the 3-D SLAM application shown in Fig. 1 to better illustrate the partitioning and orchestration framework. The 3-D SLAM application utilizes a camera, light detection and ranging (LiDAR) sensors, and inertial measurement unit (IMU) readings to rebuild the environment in a 3-D map and also uses machine learning (ML) to recognize the objects in the video. Combining the 3-D map and object recognition, this 3-D SLAM application aims for semantic maps, environment detection, and point cloud learning. This application can be partitioned into several tasks: 1) reading the video data from camera sensor; 2) reading the data from LiDAR sensor; 3) preprocessing the images; 4) object detection using the YOLO algorithm [24]; 5) 3-D map construction by using the data from the camera, LiDAR, and odometer; etc. Due to the heterogeneous nature of these tasks, the limitation of a single device, and QoS requirements, these tasks need to be distributed into multiple edge nodes. For instance, a unit control board (UCB) is used for collecting the raw data from the camera, LiDAR, and IMU and preprocessing the sensor data, but it cannot process the tasks like object detection efficiently due to insufficient GPU power. Therefore, the UCB offloads the tasks of object detection to a nearby edge node for further processing. In this case, the edge node and the UCB complement each other and collectively run the 3-D SLAM application.

B. Components of EdgeVision Framework

Fig. 2 shows the design of our proposed vision application partitioning and orchestration framework for heterogeneous edge computing platforms. The framework mainly consists of three parts: 1) *Orchestrator*; 2) *Directed acyclic graph* (DAG); and 3) *Edge cluster*.

The *Orchestrator module* is mainly responsible for partitioning the application into various tasks according to the application properties, SLA, resources profiler, and scheduling algorithm. The processing application can be made of different kinds of computing tasks. In our use case, the 3-D SLAM application contains the tasks of camera process, LiDAR process, objection detection, 3-D map construction, and coordination. The *SLA component* defines the service requirements or QoS requirements such as the application processing latency. The *resource profiler component* handles the computing and communication resources of the systems. The computing resources consist of the computing resource availability, the processor types and capacities, and costs for using such computing resources. The communication resources consist of network information, such as bandwidth, latency between each edge node, and costs for using communication resources. The *scheduling algorithm component* decides how to arrange the tasks to fulfill the SLAs under the constraints of computing and network resources and to minimize the system's overall cost.

The DAG *module* receives the scheduling tasks and topology from Orchestrator and constitutes the DAG in the data flow process format, as shown in Fig. 2. The DAG module consists of the *input component*, *data flow processes*, and the *output component*. The input component receives the input information for the tasks. The data flow processes, i.e., $S1, S2, \ldots, S7$ in the figure represent tasks partitioned from the processing application. The relation among these tasks can be parallel or sequential. These tasks are usually dependent. Therefore, if there is one task being processed at a slow speed, the whole system can be influenced.

The *Edge cluster module* consists of a master node and various kinds of slave nodes, all running KubeEdge [25] and forming a KubeEdge Cluster. The master node runs the cloudcore component from KubeEdge and is the central controller for the Edge Cluster. The slave nodes run edgecore component from KubeEdge and are equipped with heterogeneous hardware resources for computing, communication, and storage. By utilizing the orchestration feature of KubeEdge, the tasks run inside the container in different pods in these slave nodes. The master node receives the deployment information from the DAG module that consists of the topologies, all tasks, and deployment plans. The master node then further deploys the tasks into different slaves nodes in a container format managed by KubeEdge.

The above modules create a framework to partition the application and orchestrate the tasks in a heterogeneous edge computing cluster. However, mapping these tasks into the heterogeneous edge nodes graph is still a challenge, considering: 1) the devices have different runtime environments and libraries to support the tasks; 2) the edge nodes hardware

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on May 09,2022 at 08:13:51 UTC from IEEE Xplore. Restrictions apply.

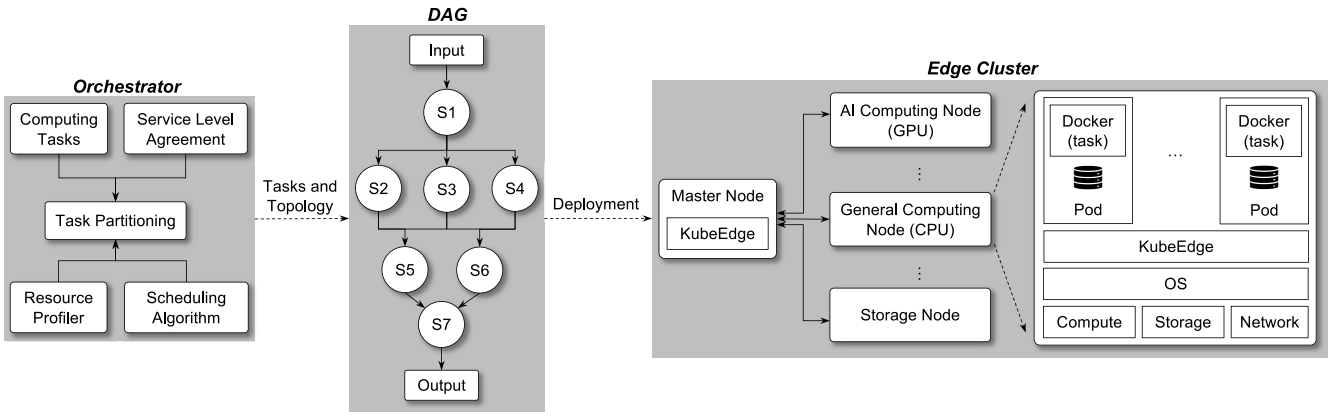

Fig. 2. EDGEVISION framework.

resources are also heterogeneous (e.g., some edge nodes are equipped with GPUs while others may not); 3) tasks are complex in terms of their resource requirements, such as CPU, GPU, memory, and network resources; and 4) various partitioning choices for the application with diverse task topologies and various task allocation strategies with a tradeoff between the SLA and resource utilization. In this section, we introduce some features of our framework in detail to deal with the first two challenges. Solutions to the last two challenges will be discussed in Sections IV and V.

C. Runtime Abstraction Feature

One important factor in edge computing and in our framework is the heterogeneity of the runtime environments. The edge nodes have a variety of runtime environments that may not be able to host and run the application tasks. Some edge nodes may lack the necessary dependencies or compatible OS to run the tasks such as video processing with ML libraries (Tensorflow or CUDA). We address this issue by leveraging container technologies, such as Docker [26]. Docker is an operational level virtualization technique that abstracts the lower layers with hardware device details. Docker offers a lightweight, and portable method to host various applications [26]. Dockers are being used in both industries and academia [27]–[29], such as in smart cities and Internet of vehicles. The work in [28] shows that a docker container can be turned on or off speedily, within near 50 ms. In our application scenario, the docker wraps different tasks with their library dependencies and OS into Docker images, which can be uploaded to a Docker repository (either in a local server or a remote server). In this case, any edge devices with the Docker engine can fetch the Docker images from the Docker repository and run the task at a local place. The users do not have to consider the low-level hardware resources and runtime environments because of Docker's feature "write once and run anywhere." There will be some differences when the applications run on Edge host and Edge container platforms. The added cost in terms of resources in this particular scenario and setup has been analyzed and discussed in a previous work [27]. In our previous work [27], a comprehensive performance evaluation of docker container-based virtualization in terms of resource usage and performance in this context has been conducted and revealed a negligible added workload compared to the added benefits in terms of flexibility and scalability. In a comparison of resource utilization between the edge host OS and the edge container with regard to executing the long short-term memory-based encoder–decoder (LSTM-ED) model, we observed sufficiently similar performance, 27.03% and 27.69% for the CPU usage and 93.84% and 94.63% for the memory one, respectively. The results suggest that, compared to running ML on the host OS, the container-based virtualization does not introduce a considerable performance downgrade but offers additional flexibility and scalability to the deployment of applications, which is a complement to the computing and intelligence features in the edge computing paradigm. For the complete details of the evaluation results, refer to our previous work [27].

D. Hardware and Resource Abstraction Feature

We further introduce the hardware and resource abstraction feature to hide the underlying hardware heterogeneity and to provide a homogeneous resource interface. As Fig. 2 shows, we utilize KubeEdge in our architecture and divide the devices into two categories: the master node and the slave nodes. According to the functionalities of KubeEdge, users can define the common hardware resource capacities (CPU, RAM, etc.) and external resource capacities (sensors, actuators, etc.) during the configuration phase. These resources are represented as resource units in Kubernetes [30], which can be represented as a YAML file during the configuration phase. Therefore, we define N tasks and E edge nodes in the system, and we define all the edge nodes with vertex $R_e^i \in R_E^N$, $(\forall 1 \leq i \leq N \ \forall 1 \leq e \leq E)$. We further define a capability description for our processing task i in edge node e with notation $R_e^i : \{R_{e,\text{cpu}}^i, R_{e,\text{gpu}}^i, R_{e,\text{ram}}^i, R_{e,\text{net}}^i, R_{e,\text{sen}}^i, R_{e,\text{other}}^i, U_e\}$, where $R_{e,\text{cpu}}^i$, $R_{e,\text{gpu}}^i$, $R_{e,\text{ram}}^i$, $R_{e,\text{net}}^i$, and $R_{e,\text{sen}}^i$ denote the CPU, GPU, RAM, network, and sensor/actuators resources situations when processing task i in edge node e, respectively. $R_{e,\text{other}}^i$ defines the other resources and can be flexibly adjusted according to the applications. U_e denotes the unit cost for using the edge node e, consisting of the unit cost for using CPU: $U_{e,\text{cpu}}$, GPU: $U_{e,\text{gpu}}$, RAM: $U_{e,\text{ram}}$, network resources: $U_{e,\text{net}}$,

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on May 09,2022 at 08:13:51 UTC from IEEE Xplore. Restrictions apply.

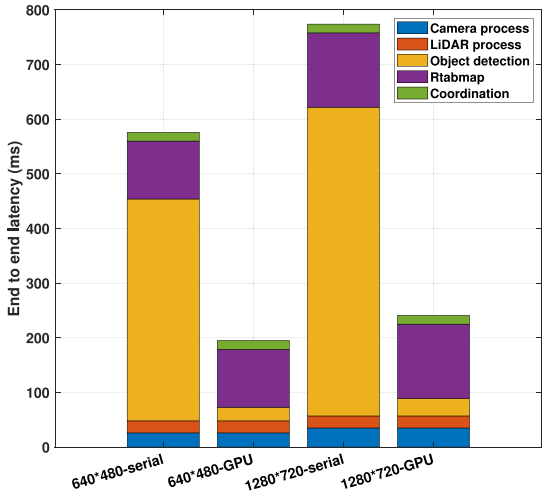

Fig. 3. Latency breakdown of the 3-D SLAM application under different configurations.

and other customized resources by edge server providers (usually from the telecommunication companies [31]). Inside the nodes, we can further divide the resources into different pods with different resource constraints. Users can regard pods in nodes as subnodes using the similar resource notation above, but the communication cost inside the nodes can be neglected.

The benefits for using such a hardware and resources abstraction feature are threefold: 1) the heterogeneous devices form a homogeneous resource pool by mapping the devices into nodes; 2) the end users can configure and adjust the device resources according to the applications need; and 3) the edge devices can easily monitor the resources and dynamically adjust the resource usages. In the next section, we formulate the scheduling problem.

IV. Problem Formulation

In this section, to formulate the problem and deploy the tasks into the EdgeVision framework, we introduce the process of preanalyzing the application and the task model.

A. Preanalysis of the Application

Before presenting the task model, we first look into the 3-D SLAM application and break it down into several tasks: camera process, LiDAR process, object detection, real-time appearance-based mapping (rtabmap), and coordination. We profile the processing latency of these tasks with different configurations in both edge nodes with CPU (Intel Core i7) and edge nodes with GPU (Jetson AGX with 512-core GPU) [32], as shown in Fig. 3. We can see that the object detection task module is the predominant bottleneck for application performance with respect to latency, which becomes worse as the video resolution increases. Furthermore, executing the object detection task module in a GPU reduces the latency significantly with more than 90% latency reduction in the case of resolution 640 × 480. By knowing the latency information of each module in the application, we can better make scheduling decisions. In the following sections, we will discuss the task model and SLAs for the application.

B. Task Model

In this section, we discuss the task model in our framework. We assume an application A that can be partitioned into N tasks in the system (our system model can be easily extended to multiple applications at the same time), with a set of tasks $\{T_1, T_2, T_3, \ldots, T_N\}$. We assume that the system optimizes the task scheduling and dispatching in each time cycle T_{cycle}. Each task T_i can be represented as $(D_I^i, D_O^i, \tau_i, \text{Reward}_i)$, where D_I^i denotes the size of data input, D_O^i means the size of data output, τ_i means the worst case execution deadline for the task T_i, and Reward_i means the reward for any edge node that hosts the task T_i. The reason for introducing the reward parameter Reward_i is to motivate edge nodes to participate in the resources exchange market and hence increase the resource usage efficiency in the system. These partitioned tasks form the application as a DAG way. In our evaluation, each task is represented as a job in Kubernetes by using the YAML script [33].

We further define the task dependency graph and edge nodes communication graph in the following.

1) Task Dependency Graph: The task dependency can be presented as a DAG: $G_{\text{task}} = (T_{\text{task}}, L_{\text{task}})$, where vertex $T_i \in T_{\text{task}}$, $(\forall 1 \leq i \leq N)$ represents all the partitioned tasks and L_{task} denotes the data communication flow among tasks: link $(T_i \rightarrow T_j) \in L_{\text{task}}$ denotes the output of task T_i is transferred as the input of T_j.

2) Edge Nodes Communication Graph: The connection of edge nodes can be expressed as a graph $G_{\text{com}} = (E_{\text{com}}, L_{\text{com}})$ where $E_e \in E_{\text{com}}$ (for $e = 1, \ldots, K$) represents the set of all edge nodes and L_{com} denotes communication link connections between edge nodes: link $(E_e, E_f) \in L_{\text{com}}$ denotes the edge nodes E_e and E_f can communicate with each other.

3) Service-Level Agreement: Given a set of tasks from the 3-D SLAM application and a set of heterogeneous edge nodes with different resources and connections, the design goal of the system is to orchestrate these tasks into the edge nodes and efficiently perform the tasks and guarantee the QoS. Each application has its own SLAs or QoS requirement. We consider an end-to-end delay in this article (the SLA can be easily extended to other metrics, which we will discuss in Section VII). The end-to-end delay for application A can be expressed as D_A, which consists of computing and communication latency from all the tasks T_{task}.

Based on the above QoS requirement, we can formulate the optimization problem as a multiobjective constrained optimization problem

$$\begin{aligned} \min \ & C_A = \sum_{e=1}^{K} C_{\text{usage}}^e \\ \min \ & D_A = \sum_{i=1}^{N} D_{\text{compu}}^i + \sum_{i=1}^{N} D_{\text{commu}}^i \\ \text{s.t.:} \ & G_{\text{task}}, G_{\text{com}} \text{ are satisfied} \end{aligned} \tag{1}$$

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on May 09,2022 at 08:13:51 UTC from IEEE Xplore. Restrictions apply.

where C_A is the total cost for processing the application A calculated as the sum of the usage cost for each edge node C^e_{usage}, consisting of CPU/GPU, RAM, and network resources usage. D_A is the total delay for processing the application A consisting of two parts, computing latency D^i_{compu} and communication latency D^i_{commu} for all the partitioned tasks.

Finally, we summarize some additional assumptions for our system model: 1) we assume that there are no malicious nodes that attempt to give false results; 2) we assume that edge nodes are willing to contribute their resources to the tasks from users for receiving incentives; 3) we assume that there are no resource contentions which will cause interruption of ongoing tasks; and 4) we assume that the edge nodes are stable during each scheduling cycle, which means that no edge node joins or leaves the system during the cycle.

We are now supporting offline allocation decisions specified in some configuration files or in the form of annotations within the mobile applications. Note that our proposed EDGEVISION framework is also suitable for other kinds of vision applications, such as robotic vision applications, augmented reality, and intelligent drone applications, because these new applications can also run on heterogeneous edge computing platforms with both CPUs and GPUs. In order to extrapolate our framework to other scenarios, we need to perform a preanalysis of the application, generate the Task Dependency Graph and the Edge Nodes Communication Graph, and define the SLA. There are some expertise needed in the process of performing a preanalysis of the application in order to decide if the applications belong to vision applications in heterogeneous edge platforms and decide the task partitioning granularity. We also need to profile the processing latency of these tasks with different configurations in both edge nodes with CPU and edge nodes with GPU because by knowing the latency information of each module in the application, we can better make the scheduling decisions. To reduce manual interactions, automatic task profiling algorithms can be designed. For automatic task allocation or online allocation decisions, we need further dynamic context-based scheduling algorithms. In this study, we limit our scope to have a real implementation for task partitioning and orchestration for heterogeneous edge platforms with offline allocation decisions. However, the automatic task profiling algorithms and dynamic context-based scheduling algorithms are part of our future works.

V. SOLUTION METHODOLOGY

Having the problem formulated, in this section, we propose the solutions of how to construct the task topology and introduce the task scheduling processes.

A. Task Topology Construction

Given the above functional explanation of the DAG module, we consider the following two practical topologies for the application of video processing when equipped with heterogeneous hardware resources (CPUs and GPUs), as shown in Fig. 4.

1) Serial DAG: In a serial DAG, we can either put all the tasks into a single node for the benefit of less communication overhead or assign the individual tasks to different nodes in a serial way, generating a pipeline flow to reduce the queuing time, as shown in Fig. 4(a). When processing images or videos, it is faster to compute the bottleneck function on a GPU than on a CPU. The nonbottleneck functions can be scheduled either on the same host node or on other remote CPUs.

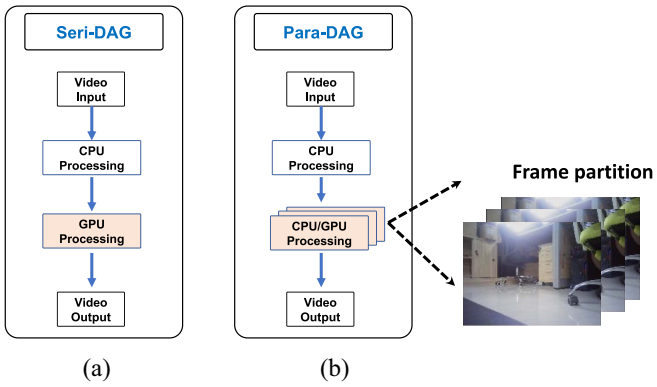

Fig. 4. Two types of task topologies: (a) serial-DAG and (b) parallel-DAG.

2) Parallel DAG: The application can also be partitioned into parallel tasks running as a DAG, leveraging both data parallelism and task parallelism. The DAG module in Fig. 2 shows a DAG-based parallel task execution model where the data flow is divided into different branches and grouped together at the output module. Fig. 4(b) shows a parallel DAG model for video applications where the video data is separated into different pieces (frames) and being processed in different branches. Depending on the applications, users can use different partitioning strategies, e.g., hash partition, composite partition, or range partition. In our 3-D SLAM application, we utilize functional module partitioning of the application and frame partitioning of the video stream. In the next section, we dig into details on how we schedule these partitioned tasks and how we deploy them.

B. Task Scheduling

How to arrange the tasks over the edge nodes according to tasks topologies is also a challenge due to the heterogeneity of tasks, edge nodes, and user requirements. We first introduce three common task schedulers from Kubernetes: 1) the least requested priority (LRP) scheduler; 2) the balanced resource allocation (BRA) scheduler; and 3) the service spreading priority (SSP). The LRP scheduler allocates the tasks to the edge nodes with maximum CPU and memory resources percentages (rather than the amounts). The BRA scheduler chooses the edge nodes with the most balancing resource with CPU and memory, which avoids all the tasks being assigned to a node consuming most of the CPU resources so as to avoid unbalanced consumption of CPU and memory resources. The SSP scheduler tries to spread the tasks within the same service to as many edge nodes as possible, to have better redundancy. However, these three schedulers do not consider the demand for resources of the tasks and how to match these tasks with the available resources while satisfying QoS requirements. These

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on May 09,2022 at 08:13:51 UTC from IEEE Xplore. Restrictions apply.

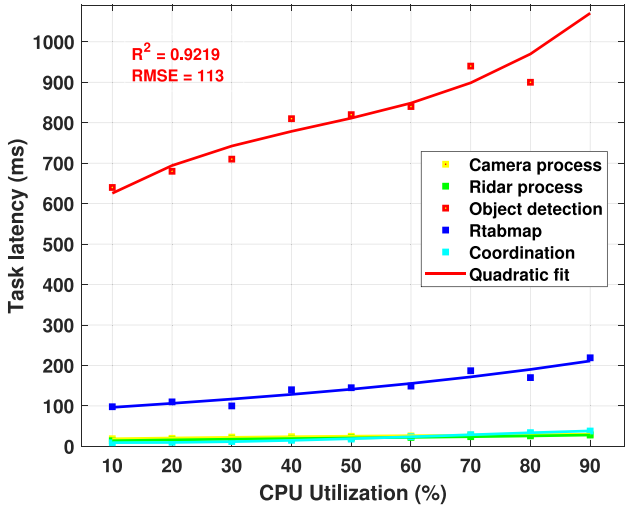

Fig. 5. Estimated processing latency versus CPU utilization.

schedulers also do not have a reliable estimation mechanism, which may also lead to resource waste by overestimating or underestimating the needed resources. Furthermore, they do not consider GPU computing resources.

The scheduler is composed of three modules to accommodate the application partitioning and QoS requirements: 1) the task resource estimation module estimates the resources requirements for executing the tasks; 2) the edge node resource monitoring module monitors the available resources for the edge nodes and can be placed in the master node (in edge server or in the Cloud); and 3) the task scheduling algorithm module deploys the tasks into edge nodes depending on the requirements of the applications or users. In our use cases, we propose two algorithms: 1) the minimum latency algorithm and 2) the minimum cost algorithm. Below we describe these modules one by one.

1) Task Resource Estimation: Before dispatching the tasks to the edge nodes, we first estimate the task performance (i.e., processing latency) and resource requirements (i.e., CPU/GPU usage, memory usage, and network usage). The task consumes the same memory and network resources no matter on which hardware device the task is executed. The memory resource consumption can be captured by the *top* utility [34] while the network resource consumption can be calculated by counting the byte array length of output stream D_O^i. The CPU/GPU usage and processing latency metrics depend not only on task properties but also on the workload of the edge nodes. In this case, we implemented a profiling phase that we execute each partitioned task on every edge node (including both CPUs and GPUs) at different workloads from 10% to 90% with an interval of 10%. Fig. 5 shows the processing latency versus the CPU workload of each module in 3-D SLAM with video resolution 1280 × 720, running on the edge node with Intel i7 processor. We use the quadratic model to fit the measuring points inspired by the work [35] which provides the satisfied predictive mean value. In this model, we use the processor CPU resource utilization and the task as input and give an estimation of the processing latency of the task.

2) Edge Node Resource Monitoring: This module periodically collects the available resources for each edge node. We use the *iPerf/scp* utility [36] to collect the port bandwidth information of the edge nodes. The CPU and memory utilization and available resources on the edge nodes are collected by the *top* utility while GPU utilization is collected by the *nvidia-smi* utility [37].

3) Task Scheduling Algorithm: For our use case, we propose two algorithms: 1) the minimum latency task scheduling (MLTS) algorithm and 2) the minimum cost task scheduling (MCTS) algorithm. Considering the different aspects of serial-DAG and parallel-DAG, we analyze both topologies when deploying the tasks. For serial-DAG, we need to analyze the bottleneck tasks and place them to the edge node with a GPU for acceleration. As shown in Fig. 1, in our use case, the object detection module using the YOLO library can actually be placed to the edge node with a GPU. The other nonbottleneck tasks will be scheduled on the edge nodes with a CPU as these tasks usually require much lower computing speed.

We first discuss how the CPU tasks are scheduled by using our proposed algorithms: MLTS and MCTS. Scheduling the GPU tasks can use similar strategies. For MLTS, given a batch of partitioned tasks to be deployed into the edge nodes, these tasks are first ranked in descending order of the required CPU computing time that can be calculated on the same processor. Then the tasks from the task list are scheduled one by one. In order to get the overall latency, we need to measure and calculate the computing latency and communication latency. The details of the algorithm MLTS are introduced in Algorithm 1. For the MCTS algorithm, the primary goal is to reduce the cost to the lowest level while satisfying the worst case execution time τ_i of the task. In this case, we filter the edge nodes which can fulfill the requirement $D^i_{e,\text{commu}} \leq \tau_i$ and choose the edge node with minimum $C^i_{e,A}$. The details of the algorithm MCTS are introduced in Algorithm 2.

4) Task Scheduling Algorithm Analysis: We further present the performance of the two algorithms, in terms of complexity (worst case performance, best case performance, and upper bound), and application scenarios.

Complexities: For Algorithm 1, the decisive variables are the number of tasks N in step 3 and the number of Edge nodes E in step 4, controlling the two For loops. The main purpose of this algorithm is to find the optimal edge node with the lowest latency in step 9, which decides the worst case performance and best case performance for the algorithm. For the worst case scenario, the algorithm needs to iterate the two For loops in order to find the optimal edge node. Therefore, from steps 4 to 8, it costs $5*N*E$ time executions while steps 3 and 9 take N executions for each step. Thus, we can estimate the complexities for Algorithm 1 in the worst case scenario: $\mathcal{W}(5*N*E+2*N)$. For the best case performance, the algorithm chooses the optimal edge node in the first trial. Thus, the complexities for Algorithm 1 in the best case scenario is $\mathcal{B}(N+E)$. From the worst case performance ($\mathcal{W}(5*N*E+2*N)$), we can obtain the upper bound for the algorithm: $\mathcal{O}(N*E)$.

Similarly, we obtain the complexities for Algorithm 2. From steps 4 to 9, it costs $6*N*E$ time executions while steps 3

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on May 09,2022 at 08:13:51 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1: MLTS Algorithm

Input: T_{task}, E_{com}, R_E^N;
Output: Tasks deployment plan for application A: P_A;
$N = numof(T_{task})$, where $T_{task} \in G_{task}$;
$E = numof(E_{com})$, where $E_{com} \in G_{com}$;
Function MLTS (T_{task}, R_E^N)**:**
  For all $i \in [1, N]$**:**
    For all $e \in [1, E]$ **:**
      Measure the edge node's CPU utilization to get $R^i_{e,cpu}$
      Use the latency estimation model in Figure 5 to estimate the processing latency $D^i_{e,compu}$
      Use the byte array length of output stream D^i_O and the network bandwidth of edge node $R^i_{e,net}$ to estimate the communication latency $D^i_{e,commu} = D^i_O / R^i_{e,net}$, $(D^i_O \in T_{task}, R^i_{e,net} \in R^N_E)$
      Calculate the total latency for this task by $D^i_{e,A} = D^i_{e,compu} + D^i_{e,commu}$
    Select the node with the lowest latency for task i: minimum $D^i_{e,A}$
end Function

Algorithm 2: MCTS Algorithm

Input: T_{task}, E_{com}, R_E^N;
Output: Tasks deployment plan for application A: P_A;
$N = numof(T_{task})$, where $T_{task} \in G_{task}$;
$E = numof(E_{com})$, where $E_{com} \in G_{com}$;
Function MCTS (T_{task}, R_E^N)**:**
  For all $i \in [1, N]$ **:**
    For all $e \in [1, E]$ **:**
      Measure the edge node's CPU utilization to get $R^i_{e,cpu}$
      Use the latency estimation model in Figure 5 to estimate the processing latency $D^i_{e,compu}$
      Use the byte array length of output stream D^i_O and the network bandwidth of edge node $R^i_{e,net}$ to estimate the communication latency $D^i_{e,commu} = D^i_O / R^i_{e,net}$,$(D^i_O \in T_{task}, R^i_{e,net} \in R^N_E)$
      Calculate the total latency for this task by $D^i_{e,A} = D^i_{e,compu} + D^i_{e,commu}$
      Calculate the total cost for this task by $C^i_{e,A} = C^i_{e,cpu} + C^i_{e,ram} + C^i_{e,net}$, where $C^i_{e,cpu} = D^i_{e,compu} * (U_{e,cpu} + U_{e,ram})$ and $C^i_{e,net} = U_{e,net} * D^i_O$
    For task i, among the edge nodes that satisfy required deadline: $D^i_{e,A} \leq \tau_i$, select the node with minimum $C^i_{e,A}$
end Function

and 10 take N executions for each step. For the worst case scenario complexities: $\mathcal{W}(6*N*E+2*N)$. For the best case performance, the algorithm chooses the optimal edge node for the first trail. Thus, the complexities for Algorithm 2 in the best case scenario are $\mathcal{B}(N+E)$. From the worst case performance ($\mathcal{W}(6*N*E+2*N)$), we can obtain the upper bound for the algorithm: $\mathcal{O}(N*E)$. As we can see, the more the number of tasks and edge nodes, the more steps the system needs to execute.

The reason for not choosing heuristic algorithms is based on our real case scenarios where the number of edge nodes and tasks will not be very large in a single application, for instance, the value of N and E are less than ten (like in our 3-D SLAM application). In this case, our algorithms can find the best optimal solution with low execution time. Furthermore, we profile the processing latency of these tasks with different configurations in both edge nodes with CPU and edge nodes with GPU. By knowing the latency information of each module in the application, we can guarantee the best optimal solutions. Therefore, our best optimal solutions are obtained from some context information (gained from engineering work) and our proposed algorithms based on real applications, rather than purely mathematical analysis. Indeed, when the value of N and E grows, we need to find a better algorithm with lower complexity, which may obtain a suboptimal solution. Our work has some limitations in scalability that deserve further research and we explain this in the Conclusion section (cf. Section VII).

VI. Evaluation

In this section, we evaluate our proposed partitioning and orchestration framework for heterogeneous edge computing in our 3-D SLAM use case. The results show that our framework can efficiently achieve significant performance gains in terms of QoS and cost-efficiency compared to the SotA baselines.

A. Evaluation Platform

We implement our framework on a real-world testbed running the 3-D SLAM application. Our testbed consists of ten heterogeneous edge devices. In particular, we use a workstation that is equipped with 3.5-GHz Dual-Core Intel Core i7 CPU and 16-GB RAM running Ubuntu 18.04 as the local server. The local server acts as the master node and runs the scheduling algorithms. This heterogeneous edge computing platform contains four Jetson AGX Xavier [32] and one Jetson Nano [38], from Nvidia (commonly used in autonomous driving, video processing, and portable computation applications), three Raspberry Pi3 Model B boards [39], and two personal computers. Fig. 6 shows the implemented hardware platform for the edge devices. These devices represent different hardware architectures, runtime environments, and hardware capacities. We summarize their parameters in Table II. All edge devices and the local server are connected via a local router.

In order to schedule and deploy the tasks to different edge devices, we follow the processes described in Section III and perform a preanalysis of the application. The preanalysis of the 3-D SLAM application is described in Sections IV-A and V-B, and the results are shown in Figs. 3 and 5. We use Docker

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on May 09,2022 at 08:13:51 UTC from IEEE Xplore. Restrictions apply.

TABLE II
HETEROGENEOUS EDGE COMPUTING TESTBED SPECIFICATIONS

Device Type	CPU	GPU	Memory	OS
Jetson AGX Xavier	8-core ARM v8.2 64-bit CPU	512-core Volta GPU with Tensor Cores	32GB LPDDR4x	Ubuntu 18.04
Jetson Nano	Quad-core ARM A57 @ 1.43 GHz CPU	128-core Maxwell	4 GB LPDDR4	Ubuntu 18.04
Pi3 Model B	1.2 GHz quad-core ARM Cortex-A53	N/A	1GB LPDDR2	Raspbian
PC one Intel	2.4GHz Intel Core i5	N/A	8GB LPDDR4	Windows 10
PC two Macbook	2.8GHz Intel Core i7	Radeon Pro 555 with 2GB memory	16GB LPDDR3	MacOS Mojave

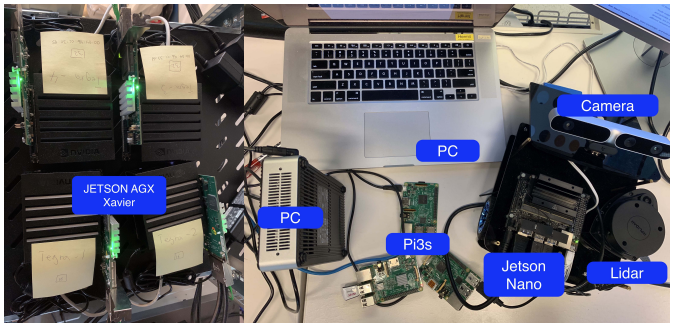

Fig. 6. Heterogeneous edge computing platform testbed.

Buildx [40] feature to create customized Dockers for our components. After that, we use our DAG module and Edge cluster module in Section IV-A to compose the tasks running on the cluster crossing the master node and edge nodes. The master node receives the deployment information from the DAG module that consists of the topologies, all tasks, and deployment plans. The master node then further deploys the tasks into different slaves nodes in a container format managed by KubeEdge. We wrap the components by a YAML script, which defines where the service is deployed, how much resource is utilized by each service, and the connection between the services and network configurations.

B. Experiments

We use the three common task schedulers mentioned in Section V-B as representative baselines: 1) LRP scheduler; 2) the BRA scheduler; and 3) the SSP scheduler. There exist some systems that are also related to harnessing heterogeneous computing resources, such as HTCondor [41], CoGTA [42], and FemtoCloud [43]. However, the homogeneous task assumption in these systems does not hold in our problem setting. Therefore, we do not include them as baselines. We choose the LRP, BRA, and SSP scheduler algorithms because they are commonly used in real industries for task scheduling. They are also chosen as baselines in other academic works as SotA algorithms [44]–[46]. Actually, these three algorithms optimize resource usage in different ways.

1) *LeastRequestedPriority:* A node is scored according to the fraction of CPU and memory (free/allocated). The node with the highest free fraction is the most preferred for the deployment. This priority function spreads the tasks across the cluster based on resource consumption. The LeastRequestedPriority strategy balances workloads based on their required resources, and the resources used among nodes are balanced.
2) *BalancedResourceAllocation:* The BRA scheduler first chooses the edge nodes with the most balancing resource with CPU and memory, which avoids all the tasks being assigned to a node consuming most of the CPU resources to avoid unbalanced consumption of CPU and memory resources. The purpose is to balance the resource allocation within each node.
3) *Service Spreading Priority:* SSP aims to ensure that the tasks of the service run on different nodes. It favors scheduling onto nodes that do not have tasks of the service already assigned there. The overall outcome is that the Service becomes more resilient to single node failures.

Therefore, comparing our algorithms with these three baselines used in industries can provide insightful information about the performance of our system on real deployments.

Our evaluation has the following three parts: 1) evaluating the scheduling algorithms for the 3-D SLAM application in edge nodes without using GPUs; 2) evaluating these schemes using edge nodes with both CPUs and GPUs; and 3) according to the two evaluations above, optimize the design for choosing CPU/GPU for this specific 3-D SLAM use case.

1) Evaluating the Scheduling Schemes Without Using GPUs: We first evaluate how different scheduling schemes handle resource heterogeneity. We only use a low frame rate, i.e., 1 frame/s, with the image resolution 1280 $\times$ 720, in order to focus on evaluating the latency performance. As Fig. 7(a) shows, MLTS performs the best with more than 30% shorter latency than all the other schemes. MLTS takes the latency of each task execution as the first priority, which leads to a shorter end-to-end delay. Instead, the other algorithms fail to consider the latency of the tasks. The primary goal of the MCTS approach is to reduce the cost while satisfying the task latencies. The MCTS approach works very similarly to the BRA approach, with a near-average 950-ms latency. The reason is that the BRA approach balances the resource allocation within each node (CPU and memory resources) while the MCTS approach is also trying to place the tasks into nodes with balanced resources in order to reduce the overall cost. Similar results are also shown in Fig. 8 when the frame rate is low. The LRP approach pushes the tasks to the edge nodes with maximum CPU and memory resources percentages and tries to balance the resource usage among nodes, with a near-average 1100-ms latency, neglecting the diverse hardware properties. For instance, a personal computer with a higher CPU utilization percentage can execute tasks faster than a Pi 3 with a lighter load. The SSP scheme splits the tasks randomly to reach a wider task deployment but causes extra communication delay with the most considerable latency near 1200 ms.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on May 09,2022 at 08:13:51 UTC from IEEE Xplore. Restrictions apply.

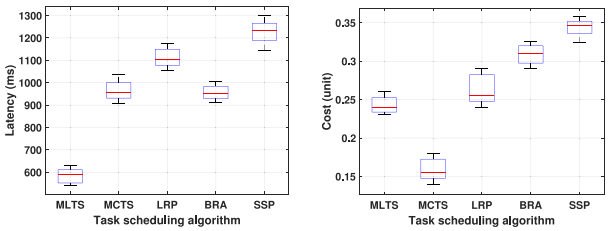

Fig. 7. (a) Comparing the latency of different tasks schedulers at a low frame rate of 1 frames/s in the heterogeneous edge platform. (b) Comparing the cost of different tasks schedulers at a low frame rate of 1 frames/s in the heterogeneous edge platform.

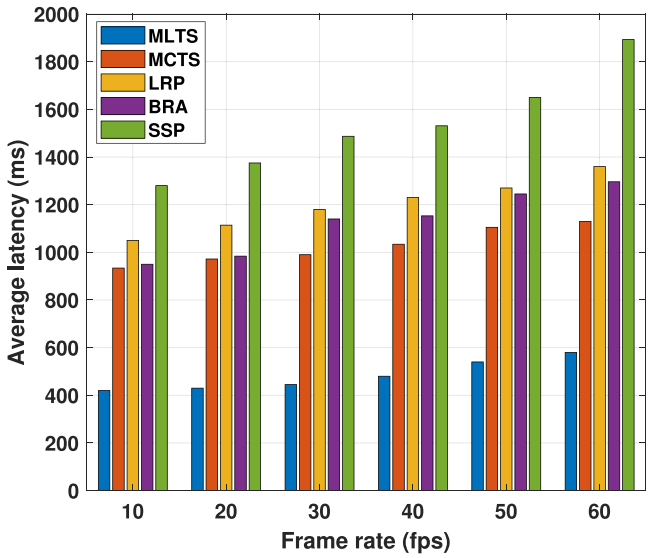

Fig. 8. Comparing the average latency for different scheduling algorithms without using GPUs.

Fig. 7(b) shows the cost for processing these tasks under different scheduling schemes. The unit cost for using different hardware resources depends on the hardware devices and we refer to Amazon EC2 on-demand pricing [47]. Fig. 7(b) shows that MCTS performs better than other schemes as the primary goal of this algorithm is to reduce the cost while satisfying the task latencies. The other schemes do not take the cost into considerations, resulting in inefficient resources utilization. Actually, an algorithm combining MLTS and MCTS can be designed according to the user's requirements. We discuss this in Section VI-B3.

We next evaluate the performance of these scheduling algorithms under different system loads. In this experiment, we have a single stream of video input and we increase the frame rate of the stream from 10 to 60 frames/s with an increasing interval of 10 frames/s. Then, we measure the average latency of the video processing. The image resolution is 640×480. The results are shown in Fig. 8. We observe that MLTS outperforms the other schedulers under different frame rates by more than 30%. In Fig. 9, we compare the average cost of different scheduling algorithms without using GPUs. We find that MCTS has the minimum cost and is less influenced by the

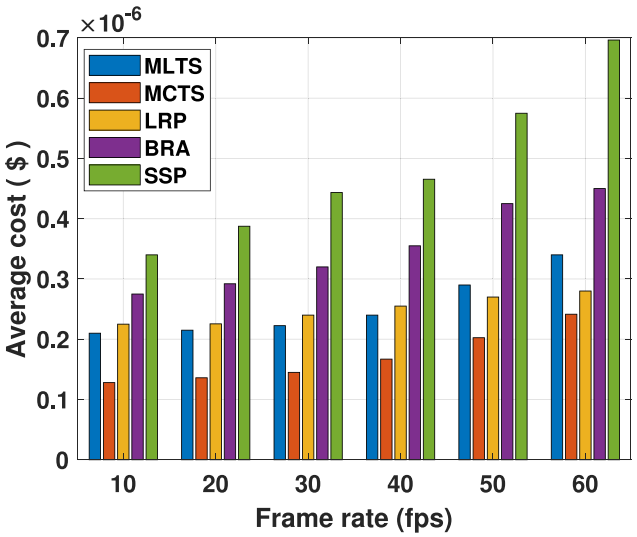

Fig. 9. Comparing the average cost for different scheduling algorithms without using GPUs.

system workloads. On average, MCTS saves more than 15% system cost than the other schemes under different system workloads. MLTS also performs better than the BRA and SSP algorithms because MLTS has less task processing latency resulting in less cost for using the edge devices. In the next section, we evaluate the performance of the schedulers on edge nodes with GPUs.

2) Evaluating the Scheduling Schemes Using Both CPUs and GPUs: Next, we consider edge nodes that have both heterogeneous CPUs and GPUs. According to our analysis in Fig. 3, the bottleneck in the 3-D SLAM application is the object detection task that consumes most of the computation. To process this object detection task, three kinds of GPUs can be utilized as shown in Table II. Fig. 10 shows the average latency of the application when applying different scheduling algorithms on edge nodes with both CPUs and GPUs. The result shows that using GPUs considerably decreases the task latency. Furthermore, the MLTS scheme outperforms the other schemes with an average latency near 240 ms. We also notice that MLTS is only little affected by the system workloads as the latter always searches for the best latency performance scheduling method. The system workloads affect the SSP schemes significantly as SSP tries to deploy the tasks to as many edge nodes as possible, which increases the system communication delay and inefficiently makes use of the GPU nodes. We further compare the average cost of the scheduling algorithms using both CPUs and GPUs in Fig. 11. We observe that MCTS has the least average cost compared to the baselines. The results further demonstrate the effectiveness of MCTS for saving the system cost while meeting the QoS requirements of the application. Furthermore, MLTS has the highest cost among all the schemes when the frame rate is higher than 20. To guarantee the lowest latency performance, the MLTS scheduler first chooses the GPUs to have the best performance, which results in higher cost in the system.

3) Optimize Design: After evaluating each technique in different settings, we highlight some observations from

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on May 09,2022 at 08:13:51 UTC from IEEE Xplore. Restrictions apply.

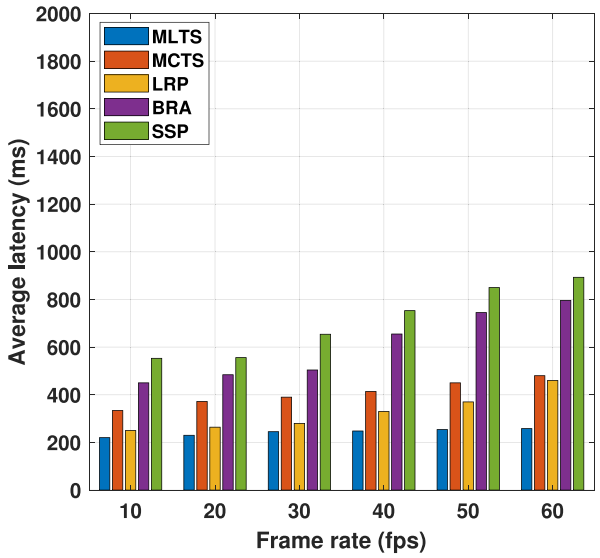

Fig. 10. Comparing the average latency for different scheduling algorithms using both CPUs and GPUs.

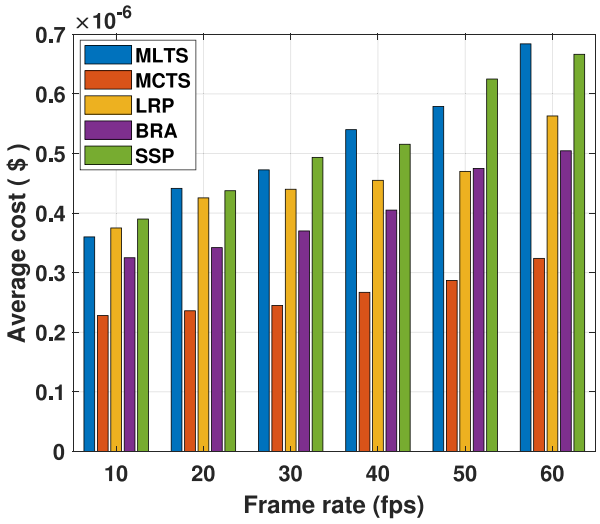

Fig. 11. Comparing the average cost for different scheduling algorithms using both CPUs and GPUs.

the results. First, suggested by the results reported in Figs. 8 and 10, we conclude that employing GPU for vision applications brings significant performance improvements in latency without increasing the cost too much. We can adopt the policy described in Fig. 12, to speed up the decision process than purely going through our MLTS and MCTS schedulers.

Second, we noticed that depending on user requirements, it is possible to combine MLTS and MCTS to provide better scheduling performance for the system. We define a new cost function considering both the system latency and resource monetary cost

$$K^i_{e,A} = \lambda_t * D^i_{e,A} + \lambda_e * C^i_{e,A}. \tag{2}$$

Note that λ_t represents the weight of latency and λ_e represents the weight of resource cost, with constraint $\lambda_t + \lambda_e = 1, \lambda_t, \lambda_e \in [0, 1]$. To improve the flexibility of the model, we

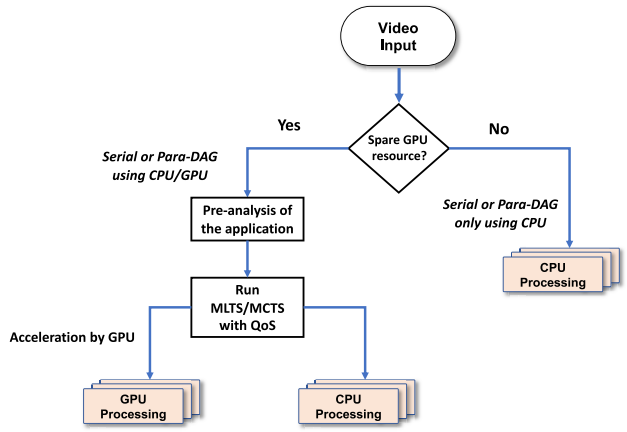

Fig. 12. Optimize the decision process for vision application partitioning.

set a constraint for λ_t and λ_e so that we can adjust the weights according to different system requirements. In practice, we can apply multiple criteria decision making and multiattribute utility theory [48] to decide about appropriate weights. To evaluate this new scheduler, we use the new cost function $K^i_{e,A}$ to replace $C^i_{e,A}$ step 10 in Algorithm 2. In this step, this new scheduler will choose the edge node with minimum cost $K^i_{e,A}$ for deploying task i considering both the latency and resource cost while still satisfying the deadline τ_i. Note that the units of latency and resource monetary cost are different and we can normalize the values of both latency and monetary cost into range [0, 1]. We normalize the latency as follows:

$$\text{normalized } \left(D^i_{e,A}\right) = \frac{D^i_{e,A}}{D^i_{\max,A}}, 1 \le i \le N \tag{3}$$

where $D^i_{\max,A}$ is the maximum latency for running task i in the edge clusters. We can also have

$$\text{normalized } \left(C^i_{e,A}\right) = \frac{C^i_{e,A}}{C^i_{\max,A}}, 1 \le i \le N \tag{4}$$

where $C^i_{\max,A}$ is the maximum resource cost for running task i in the edge clusters. For convenience, we set $\lambda_t = 0.5$ and $\lambda_e = 0.5$ in our system and evaluate this new scheduler.

Fig. 13 shows the normalized total cost of different scheduling algorithms for the new scheduler MLTS/MCTS. The normalized total cost includes both the latency and resource monetary cost. We observe the scheduler combining both MLTS and MCTS outperforms the other schemes, because the new scheduler considers the properties of both the tasks and the heterogeneous edge nodes, matching them in an optimal way.

Fig. 14 shows the normalized latency of different scheduling algorithms for the new scheduler MLTS/MCTS. We observe that the scheduler combining both MLTS and MCTS causes less latency than other approaches except for the LRP approach when the frame rate is less than 30 frames/s. This is because the LRP approach chooses CPU/GPU with the highest free fraction for the deployment without considering the resource monetary cost. However, when the frame rate increases and

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on May 09,2022 at 08:13:51 UTC from IEEE Xplore. Restrictions apply.

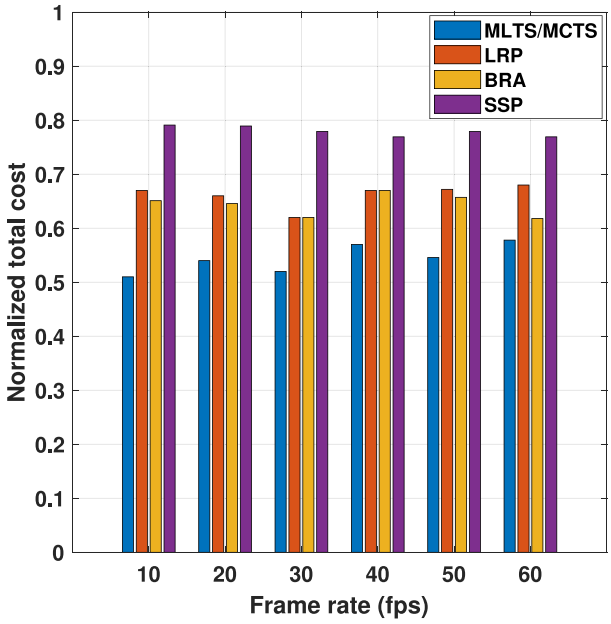

Fig. 13. Comparing the normalized total cost for different scheduling algorithms using both CPUs and GPUs.

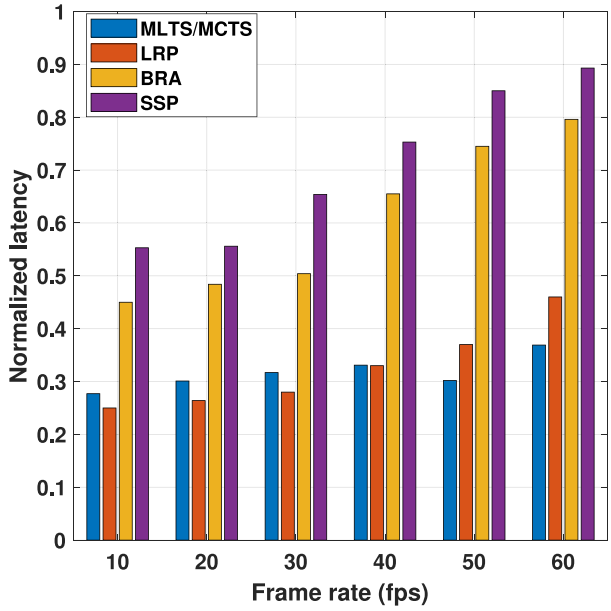

Fig. 14. Comparing the normalized latency for different scheduling algorithms using both CPUs and GPUs.

the edge platforms are under heavy load, our scheduler MLTS/MCTS taking latency into consideration outperforms LRP approaches.

VII. Conclusion and Future Work

In this article, we proposed an application partitioning and orchestration framework, called EdgeVision, for video processing applications on heterogeneous edge computing platforms. EdgeVision abstracts the heterogeneous hardware resources and the runtime environments, considering both CPU and GPU computing resources. We devised two scheduling algorithms: 1) MLTS and 2) MCTS, which, when combined, can proactively minimize the overall latency and system cost in heterogeneous edge computing platforms. We evaluated our framework and scheduling algorithms on an edge-based 3-D SLAM application in a real testbed consisting of ten heterogeneous edge devices. The result shows that our scheduling algorithms can efficiently reduce more than 30% of the task processing latency and 15% of the system cost, compared with other scheduling schemes on heterogeneous edge clusters.

Our work has some limitations that deserve further research. First, we assume the edge cluster is stable and cooperative (e.g., no churn). However, this assumption may not hold in some scenarios, such as autonomous driving. This issue can be mitigated by adding dynamic scheduling algorithms leveraging AI. Currently, EDGEVISION supports offline allocation decisions specified in some configuration files or in the form of annotations within the mobile applications. For automatic task profiling and dynamic online allocation decisions for task distribution, we need further dynamic context-based scheduling algorithms. Second, our current experiment platform consists of a limited number of edge devices and the scalability of EdgeVision needs further investigation. This issue can be handled by performing additional simulation studies using simulators.

References

[1] Y. Wu, K. Zhang, and Y. Zhang, "Digital twin networks: A survey," *IEEE Internet Things J.*, vol. 8, no. 18, pp. 13789–13804, Sep. 2021.

[2] K. Zhang, J. Cao, and Y. Zhang, "Adaptive digital twin and multiagent deep reinforcement learning for vehicular edge computing and networks," *IEEE Trans. Ind. Informat.*, vol. 18, no. 2, pp. 1405–1413, Feb. 2022.

[3] X. Yu, Y. Chu, F. Jiang, Y. Guo, and D. Gong, "SVMs classification based two-side cross domain collaborative filtering by inferring intrinsic user and item features," *Knowl. Based Syst.*, vol. 141, pp. 80–91, Feb. 2018.

[4] X. Huang, R. Yu, J. Kang, Y. He, and Y. Zhang, "Exploring mobile edge computing for 5G-enabled software defined vehicular networks," *IEEE Wireless Commun.*, vol. 24, no. 6, pp. 55–63, Dec. 2017.

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

[6] X. Huang, S. Leng, S. Maharjan, and Y. Zhang, "Multi-agent deep reinforcement learning for computation offloading and interference coordination in small cell networks," *IEEE Trans. Veh. Technol.*, vol. 70, no. 9, pp. 9282–9293, Sep. 2021.

[7] S. Mao, J. Wu, L. Liu, D. Lan, and A. Taherkordi, "Energy-efficient cooperative communication and computation for wireless powered mobile-edge computing," *IEEE Syst. J.*, early access, Oct. 15, 2020, doi: 10.1109/JSYST.2020.3020474.

[8] *Openfog Consortium Technical Report V.1.0*, OpenFog Ref. Archit., Fremont, CA, USA, 2017.

[9] S. Mao *et al.*, "Computation rate maximization for intelligent reflecting surface enhanced wireless powered mobile edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 70, no. 10, pp. 10820–10831, Oct. 2021.

[10] S. Wang, D. Ye, X. Huang, R. Yu, Y. Wang, and Y. Zhang, "Consortium blockchain for secure resource sharing in vehicular edge computing: A contract-based approach," *IEEE Trans. Netw. Sci. Eng.*, vol. 8, no. 2, pp. 1189–1201, Apr.–Jun. 2021.

[11] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo, "A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters," in *Proc. 14th USENIX Symp. Oper. Syst. Des. Implement. (OSDI)*, 2020, pp. 463–479.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on May 09,2022 at 08:13:51 UTC from IEEE Xplore. Restrictions apply.

[12] J. Liu, E. Ahmed, M. Shiraz, A. Gani, R. Buyya, and A. Qureshi, "Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions," *J. Netw. Comput. Appl.*, vol. 48, pp. 99–117, Feb. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804514002161
[13] E. Cuervo *et al.*, "MAUI: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst. Appl. Serv.*, 2010, pp. 49–62.
[14] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic execution between mobile device and cloud," in *Proc. 6th Conf. Comput. Syst.*, 2011, pp. 301–314.
[15] S.-H. Hung, C.-S. Shih, J.-P. Shieh, C.-P. Lee, and Y.-H. Huang, "Executing mobile applications on the cloud: Framework and issues," *Comput. Math. Appl.*, vol. 63, no. 2, pp. 573–587, 2012.
[16] H. Wu, W. J. Knottenbelt, and K. Wolter, "An efficient application partitioning algorithm in mobile environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 7, pp. 1464–1480, Jul. 2019.
[17] J. Cao, L. Yang, and J. Cao, "Revisiting computation partitioning in future 5G-based edge computing environments," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2427–2438, Apr. 2019.
[18] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, and A. Kitazawa, "FogFlow: Easy programming of IoT services over cloud and edges for smart cities," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 696–707, Apr. 2018.
[19] Y. Han, S. Shen, X. Wang, S. Wang, and V. Leung, "Tailored learning-based scheduling for kubernetes-oriented edge-cloud system," 2021, *arXiv:2101.06582*.
[20] Y. Yang, Z. Liu, X. Yang, K. Wang, X. Hong, and X. Ge, "POMT: Paired offloading of multiple tasks in heterogeneous fog networks," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8658–8669, Oct. 2019.
[21] R. Kumar *et al.*, "Coding the computing continuum: Fluid function execution in heterogeneous computing environments," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, Portland, OR, USAs, 2021, pp. 66–75.
[22] A. Barak, T. Ben-Nun, E. Levy, and A. Shiloh, "A package for OpenCL based heterogeneous computing on clusters with many GPU devices," in *Proc. IEEE Int. Conf. Clust. Comput. Workshops Posters (CLUSTER WORKSHOPS)*, Heraklion, Greece, 2010, pp. 1–7.
[23] R. Kobayashi, N. Fujita, Y. Yamaguchi, A. Nakamichi, and T. Boku, "GPU-FPGA heterogeneous computing with OpenCL-enabled direct memory access," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, Rio de Janeiro, Brazil, 2019, pp. 489–498.
[24] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," 2020, *arXiv:2004.10934*.
[25] "KubeEdge." 2021. [Online]. Available: https://kubeedge.io/en/
[26] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," *Linux J.*, no. 239, p. 2, 2014.
[27] Y. Liu, D. Lan, Z. Pang, M. Karlsson, and S. Gong, "Performance evaluation of containerization in edge-cloud computing stacks for industrial applications: A client perspective," *IEEE Open J. Ind. Electron. Soc.*, vol. 2, pp. 153–168, Feb. 2021.
[28] X. Huang, R. Yu, S. Xie, and Y. Zhang, "Task-container matching game for computation offloading in vehicular edge computing and networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 10, pp. 6242–6255, Oct. 2021.
[29] X. Huang, P. Li, and R. Yu, "Social welfare maximization in container-based task scheduling for parked vehicle edge computing," *IEEE Commun. Lett.*, vol. 23, no. 8, pp. 1347–1351, Aug. 2019.
[30] "Kubernetes Manage Resource." 2021. [Online]. Available: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
[31] X. Sun and N. Ansari, "EdgeIoT: Mobile edge computing for the Internet of Things," *IEEE Commun. Mag.*, vol. 54, no. 12, pp. 22–29, Dec. 2016.
[32] "Jetson AGX Xavier Developer Kit." 2021. [Online]. Available: https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit
[33] "Kubernetes Jobs." 2021. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/job/
[34] "Top Utility Tool." 2021. [Online]. Available: https://man7.org/linux/man-pages/man1/top.1.html
[35] Q. Nguyen, P. Ghosh, and B. Krishnamachari, "End-to-end network performance monitoring for dispersed computing," in *Proc. Int. Conf. Comput. Netw. Commun. (ICNC)*, Maui, HI, USA, 2018, pp. 707–711.
[36] "iPerf Utility Tool." 2021. [Online]. Available: https://iperf.fr/iperf-download.php
[37] "NVIDIA System Management Interface." 2021. [Online]. Available: https://iperf.fr/iperf-download.php
[38] "Jetson Nano Developer Kit." 2021. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit
[39] "Raspberry Pi 3 Model B+ Developer Kit." 2021. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/
[40] "Docker Buildx." 2020. [Online]. Available: https://docs.docker.com/engine/reference/commandline/buildx (accessed Mar. 28, 2021).
[41] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor—A hunter of idle workstations," Dept. Comput. Sci., Univ. Wisconsin-Madison, Madison, WI, USA, Rep. #730, 1987.
[42] D. Zhang, Y. Ma, C. Zheng, Y. Zhang, X. S. Hu, and D. Wang, "Cooperative-competitive task allocation in edge computing for delay-sensitive social sensing," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Seattle, WA, USA, 2018, pp. 243–259.
[43] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *Proc. IEEE 8th Int. Conf. Cloud Comput.*, New York, NY, USA, 2015, pp. 9–16.
[44] W. Zhang *et al.*, "URSA: Precise capacity planning and fair scheduling based on low-level statistics for public clouds," in *Proc. 49th Int. Conf. Parallel Process. (ICPP)*, 2020, pp. 1–11.
[45] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards network-aware resource provisioning in kubernetes for fog computing applications," in *Proc. IEEE Conf. Netw. Softw. (NetSoft)*, Paris, France, 2019, pp. 351–359.
[46] Z. Wang, H. Liu, L. Han, L. Huang, and K. Wang, "Research and implementation of scheduling strategy in kubernetes for computer science laboratory in universities," *Information*, vol. 12, no. 1, p. 16, 2021.
[47] "Amazon EC2 On-Demand Pricing." 2021. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/
[48] H. Wang, R. P. Liu, W. Ni, W. Chen, and I. B. Collings, "VANET modeling and clustering design under practical traffic, channel and mobility conditions," *IEEE Trans. Commun.*, vol. 63, no. 3, pp. 870–881, Mar. 2015.

Dapeng Lan (Member, IEEE) received the B.Eng. degree in microelectronics from Sun Yat-sen University, Guangzhou, China, in 2014, the first M.Sc. degree in ICT innovation from KTH Royal Institute of Technology, Stockholm, Sweden, in 2016, and the second M.Sc. degree in innovation in information and communication technology from the Technical University of Berlin, Berlin, Germany, in 2017. He is currently pursuing the Ph.D. degree in fog computing with the Department of Informatics, University of Oslo, Oslo, Norway.

In 2017, he worked in a company called InnoEnergy about smart building energy management in Sweden. He was a thesis student with ABB Corporate Research Center, Västerås, Sweden, from January to September 2016. His research interests include edge/fog computing, Internet of Things, and distributed systems.

Amir Taherkordi (Member, IEEE) received the Ph.D. degree from the Informatics Department, University of Oslo (UiO), Oslo, Norway, in 2011.

He is an Associate Professor with the Department of Informatics, UiO. After completing his Ph.D. studies, he joined Sonitor Technologies, Oslo, as a Senior Embedded Software Engineer. From 2013 to 2018, he was a Researcher with the Networks and Distributed Systems Group, Department of Informatics, UiO. He has so far published several articles in high-ranked conferences and journals, and he has experience from several national (Norwegian Research Council) and international (European research funding agencies) research projects. His research interests are broadly on resource efficiency, scalability, adaptability, dependability, mobility, and data intensiveness of distributed systems designed for emerging computing technologies, such as Internet of Things, fog/edge/cloud computing, and cyber–physical systems.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on May 09,2022 at 08:13:51 UTC from IEEE Xplore. Restrictions apply.

Frank Eliassen is a Professor and the Leader of the Networks and Distributed Systems Group, Department of Informatics, University of Oslo, Oslo, Norway. He is an Experienced Researcher and a Project Manager for several decades, in the areas of distributed systems, middleware, and IoT/cyber-physical systems (CPSs) with experience from national and EU level projects. He is also leading a strategic research activity on Energy Informatics with the University of Oslo. His present research interests include service-oriented IoT/edge/fog computing and CPS middleware and programming models in application areas, including smart cities and smart grids, adaptive software systems, autonomic systems (self-*), peer-to-peer systems, and cooperative microgrids.

Lei Liu (Member, IEEE) received the B.Eng. degree in communication engineering from Zhengzhou University, Zhengzhou, China, in 2010, and the M.Sc. and Ph.D. degrees in communication engineering from Xidian University, Xi'an, China, in 2013 and 2019, respectively.

From 2013 to 2015, he was employed a subsidiary of China Electronics Corporation, Beijing, China. From 2018 to 2019, he was supported by the China Scholarship Council to be a visiting Ph.D. student with the University of Oslo, Oslo, Norway. He is currently a Lecturer with the Department of Electrical Engineering and Computer Science, Xidian University. His research interests include vehicular ad hoc networks, intelligent transportation, mobile-edge computing, and Internet of Things.

Stéphane Delbruel received the Ph.D. degree in computer science in the field of distributed systems from IRISA—Université de Rennes 1, Rennes, France, in 2017.

He was an Industrial Research Engineer in embedded architectures and radiocommunications from 2005 to 2013, and is currently an Associate Professor with the LaBRI Laboratory, Université de Bordeaux, Talence, France. His research interests include distributed systems, edge computing, wireless sensor networks, and cyber–physical systems.

Dr. Delbruel received three Best Paper Awards, one in the field of distributed systems at IC2E 2016, a second in the field of wireless sensors networks at MobiQuitous 2017, and a third one for its work on cyber–physical systems at DEBS 2019.

Schahram Dustdar (Fellow, IEEE) is a Full Professor of Computer Science heading the Research Division of Distributed Systems, TU Wien, Vienna, Austria.

Dr. Dustdar is a recipient of the ACM Distinguished Scientist Award in 2009, the ACM Distinguished Speaker Award in 2021, the IBM Faculty Award in 2012, and an Elected Member of the Academia Europaea: The Academy of Europe, where he is the Chairman of the Informatics Section. He is the Founding Co-Editor-in-Chief of the new *ACM Transactions on Internet of Things* as well as the Editor-in-Chief of *Computing* (Springer). He is an Associate Editor of IEEE TRANSACTIONS ON SERVICES COMPUTING, IEEE TRANSACTIONS ON CLOUD COMPUTING, *ACM Transactions on the Web*, and *ACM Transactions on Internet Technology*, as well as on the editorial board of IEEE INTERNET COMPUTING and IEEE COMPUTER. In 2021, he was an Elected Fellow and the President for the Asia–Pacific Artificial Intelligence Association.

Yang Yang (Fellow, IEEE) received the B.S. and M.S. degrees in radio engineering from Southeast University, Nanjing, China, in 1996 and 1999, respectively, and the Ph.D. degree in information engineering from the Chinese University of Hong Kong, Hong Kong, in 2002.

He is currently a Full Professor with the School of Information Science and Technology, the Master of Kedao College, and the Director of Shanghai Institute of Fog Computing Technology, ShanghaiTech University, Shanghai, China. He is also an Adjunct Professor with the Department of Broadband Communication, Peng Cheng Laboratory, Shenzhen, China, as well as a Senior Consultant with Shenzhen SmartCity Technology Development Group, Shenzhen. Before joining ShanghaiTech University, he has held faculty positions with the Chinese University of Hong Kong; Brunel University London, Uxbridge, U.K.; University College London, London, U.K., and SIMIT, CAS, Shanghai. His research interests include 5G/6G, computing networks, service-oriented collaborative intelligence, IoT applications, and advanced testbeds and experiments. He has published more than 300 papers and filed more than 80 technical patents in these research areas.

Dr. Yang was the Chair of the Steering Committee of Asia–Pacific Conference on Communications from 2019 to 2021. He is a General Co-Chair of the IEEE DSP 2018 Conference and a TPC Vice-Chair of the IEEE ICC 2019 Conference.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on May 09,2022 at 08:13:51 UTC from IEEE Xplore. Restrictions apply.