IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 15 September 2023, accepted 25 September 2023, date of publication 2 October 2023, date of current version 4 October 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3321274

RESEARCH ARTICLE

Benchmarking Container Technologies on ARM-Based Edge Devices

SHAHIDULLAH KAISER[1], ALI ŞAMAN TOSUN[2], AND TURGAY KORKMAZ[1]
[1]Department of Computer Science, The University of Texas at San Antonio, San Antonio, TX 78249, USA
[2]Department of Mathematics and Computer Science, The University of North Carolina at Pembroke, Pembroke, NC 28372, USA

Corresponding author: Shahidullah Kaiser (shahidullah.kaiser@my.utsa.edu)

This work was supported by the Army Research Office under Grant W911NF-23-1-0187.

ABSTRACT Internet-of-Things (IoT) devices continuously gather data and send the data to the cloud for further processing. However, with the recent trend of increasing number of IoT devices, billions of devices are anticipated to send data to the cloud, eventually impacting performance and cost. To solve this problem, one way is to process data locally inside edge nodes. The edge nodes are closer to the IoT devices and improve the overall performance of the system by distributing cloud tasks in edge devices. Edge devices are generally resource-constrained with limited RAM, CPU, and storage. Container technologies are ideal in edge nodes due to their isolation and being lightweight. A benchmarking scheme for containers on edge devices can help compare container technologies, hardware devices and architectures, and software. However, there is not sufficient research in this direction. So, in this work, we take a step towards developing this benchmark. We explore and evaluate the performance, efficiency, and suitability of different container technologies, including Docker, Podman, and Singularity, in the context of edge computing on ARM-based devices. Our experiments include evaluating computer vision applications that employ Haar Cascades, HOG, CNN with YOLO algorithm, and data science workloads commonly encountered in edge computing scenarios. We devised sets of performance metrics to assess container technology, including waiting time, receiving time, processing time, resource utilization, and throughput. Besides, we investigate how different container technologies optimize resource utilization and compare their efficiency on ARM-based edge devices. Our benchmarking analysis yielded valuable insights into the strengths and limitations of each container technology. Our results reveal that Docker, Podman, and Singularity containers exhibit diverse resource consumption patterns and network efficiency. Docker container has better CPU and RAM utilization for most applications. Docker also boasts the lowest waiting time of approximately 0.9 seconds, comparable to native performance. In terms of processing time, Docker excels in Car detection (0.12 seconds), while Singularity and Podman outperform Docker in Object detection. Notably, native systems exhibit a remarkable improvement over containers (average of 1.2 seconds) in Object detection (0.98 seconds), highlighting the challenges of resource-intensive deep neural network algorithms on edge devices. Based on our findings, we offer practical advice for picking the best container technology for specific use cases in ARM-based edge computing. We also offer a set of benchmarking approaches and metrics that can be used to drive future research on container technologies on ARM platforms.

INDEX TERMS Edge computing, container, Docker, singularity, Podman, Internet of Things (IoT), benchmark, computer vision, data science.

I. INTRODUCTION

The Internet of Things (IoT) encompasses a wide network of interconnected smart objects implanted with sensors, processing capabilities, and networking functionality, enabling advanced features. Millions of these sensors generate and gather substantial amounts of data, necessitating data transmission across the network to enable their operation. However, the computing capability of IoT devices, comprising storage, processing, and networking resources,

The associate editor coordinating the review of this manuscript and approving it for publication was Christos Anagnostopoulos.

is insufficient to conduct compute-intensive activities locally. The increased number of connected devices generates a large volume of data, posing another problem for today's networks to handle adequately [1], [2]. Edge computing is a feasible option to classify and filter large amounts of IoT data before transmitting them to the cloud center [3]. Edge computing enables processing compute-intensive operations on resource-constrained IoT devices that cannot be completed locally [4], [5]. Besides, edge computing has the potential to cut network expenses, reduce transmission delays, prevent service failures, avoid bandwidth limits, and provide better control over sensitive data transfer. Moreover, load times are reduced, and deploying online services closer to users enables both dynamic and static caching possibilities. Research shows that utilizing edge computing can enhance response times by 80-200 milliseconds and decrease energy consumption by up to 40% for certain applications [6], [7].

Container technology is gaining popularity in IoT and Edge Computing due to its portability, faster deployment, lightweight and easier management [8]. Sensors and embedded devices in the IoT collect huge amounts of data from the natural environment or industry and transfer it to public or private cloud servers [9], [10]. Besides, these IoT devices employ lightweight operating systems and software that require frequent maintenance and protection. Containers are in handy in this circumstance. They enable developers to design an application and execute it on any platform that supports containers without concerns about dependencies or compatibility issues. This simplifies the deployment and management of applications in distributed contexts such as IoT devices and edge servers. Containers also provide advantages such as enhanced resource use, faster deployment times, and easier application scaling. Moreover, they help to improve security by isolating apps from one another and the underlying system, ensuring enhanced protection for sensitive data. In summary, containers serve as a valuable tool for developing and deploying applications in IoT and edge computing contexts, addressing the challenges of data management, maintenance, and security while facilitating efficient and scalable application deployment.

Containerization has started gaining popularity on the ARM architecture to utilize hardware resources better and enhance energy efficiency [11], [12]. Companies have started deploying containers and well-known cloud vendors provide container services for ARM architecture, such as Amazon Fargate [13] and Microsoft Azure Kubernetes [14]. Besides, containerization in ARM-based edge devices is a popular area of study. Rancher lab [15] is creating AI-enabled infrastructure by merging rugged, low-power ARM devices with K3 in order to run small Kubernetes clusters that effectively transport data. The lab ensures that leveraging ARM not only enhances connectivity but also can save costs. Their goal is to widely deploy K3 as container orchestration on edge, and ARM is projected to be the edge leader in the near future. Containers being compatible with both single-board devices and clusters makes it well-suited for building cloud-based Edge device clusters. Several academic and scientific institutes have created ARM-based Raspberry Pi clusters including 9-node TinyTitan [16] by Oakridge National Lab's, Online RPi simulator [17] by Microsoft, and 32-node RPi cluster project by Boise State University [18]. Edge-Cloud is created by combining Cloud and IoT, which necessitates constrained hardware and lightweight virtualization. Containerization is a solution to this lightweight virtualization, and containers are especially important for cloud computing as Platform-as-a-Service (PaaS). One of the requirements for edge cloud PaaS are cost-efficiency, lower power consumption, and robustness [19]. Implementing containers on Raspberry Pi can fulfill these requirements efficiently.

The fundamental motivation for writing this article is to benchmark and assess the key container tools for the edge domain to assist in selecting appropriate container technology for edge computing solutions. We used four computer vision applications, e.g., Face Detection (Haar Cascades Algorithm), Vehicle Detection (Haar Cascades Algorithm), Body Detection (HOG Algorithm), and Object Detection Algorithm (CNN by YOLO) to benchmark the container technologies studied in this study. Besides, we use Fitness Tracker data to evaluate the data science workload. The goal is to provide insights into the advantages and disadvantages of different container technologies in Computer Vision Applications and Data Science and identify the best solutions for specific use cases. We also aim to quantify the performance impact of using container virtualization technology compared to native execution and to evaluate the security and isolation of container-based systems. The benchmarking results can help users make informed decisions when selecting container technologies for edge computing and specific applications.

The contributions of the paper are as follows:

- The study reviews prominent container technologies such as Docker, Podman, and Singularity on ARM-based edge devices. We provide insights into the feasibility of each container technology for edge computing environments by comparing their performance and resource consumption characteristics.
- Instead of relying on synthetic benchmark tools which do not accurately reflect IoT-Edge scenarios, we developed a set of applications to benchmark image data and stream data performance.
- Conducting a detailed performance benchmarking analysis of the containers. We measure key performance indicators such as receiving time, waiting time, processing time, memory, and CPU usage. This benchmarking enables a quantitative comparison of the containers' efficiency and performance in handling image data processing tasks.
- Explore the utilization of data science workloads, such as Fitbit fitness tracker data processing, to evaluate the performance of containers for stream data. The paper provides a broader perspective on the containers' performance in handling diverse workloads in edge computing

scenarios by considering CPU usage, sending time, and memory usage for data science workloads.
- Highlight the strengths and limitations of each container technology, aiding developers and practitioners in selecting the most suitable container technology for specific edge computing use cases.

We have made all the sender, receiver script, and implementation of algorithms available in [20]. The researchers can utilize them for their research.

The rest of the paper is organized as follows: Section II presents the review of related work and Section III discusses background study. Section IV introduces the proposed scheme. We explained our benchmark design and benchmarking criteria in V. The results are presented in Section VI. Section VII discusses the important aspects of the results. The paper concludes in section VIII and discusses future works.

II. RELATED WORK

The usability of containers at the edge has been well documented. Research work [21] explores and evaluates the performance of Docker and Containered runtimes as well as Kubernetes and Docker Swarm container orchestration tools for IoT devices. The authors conducted experiments using synthetic benchmarking tools to test the performance of CPU, memory, disk, and network to evaluate the performance of containers. The results showed that container technology suits low-power and resource limited IoT devices.

Morabito conducted a complete performance evaluation to demonstrate the advantages and disadvantages of various low-power devices when managing container virtualization [22], [23]. In their experiment, they used five different devices: a Raspberry Pi 2 Model B, a Raspberry Pi 3 Model B, an Odroid C1+, an Odroid C2, and an Odroid XU4. The devices were all built on ARM-based single-board processors. They only use Docker container and Native to measure CPU, power consumption, memory, network, and disk performance. The temperature of the board was also measured. The results demonstrated that adopting container virtualization technology on single-board computers (SBC) had no performance impact compared to native execution.

Buyya and Srirama [24] evaluated a lightweight container middleware for edge cloud architectures. They used microservices and container-based clustered environments to address limitations in orchestration and DevOps in the cloud context. They also discussed using blockchain technology for provenance management and other security concerns. The final outcome of their work is the demonstration that containers are the most suitable technology platform for an edge cloud PaaS, and the proposal of a fourth-generation PaaS that bridges the gap between IoT and cloud technology.

A performance benchmarking of containers, hypervisor, and unikernel was done for ARM and x86 in [25]. They selected Docker as the container, KVM as the hypervisor, and Rumprun and OSv as unikernels. The evaluation tools were synthetic benchmarks, i.e., Sysbench (CPU), Iperf (Networking), and STREAM (Memory). The results demonstrated that containers and unikernels perform slightly better than a hypervisor for CPU and memory-related workloads on x86 architecture.

Recent research efforts have extended into benchmarking studies that assess technology performance and robustness within specific application domains. Notably, a study in the field of autonomous driving presents a comprehensive benchmark on the corruption robustness of 3D object detection models [26]. This study evaluates the performance of various models across different corruption types on datasets from KITTI, nuScenes, and Waymo. The findings provide insights into the effects of corruptions, the comparative robustness of different models, and the efficacy of data augmentation strategies. The correlation between corruption robustness and clean accuracy, as well as the trade-off between robustness under image and point cloud corruptions, is thoroughly explored. This benchmark study not only contributes valuable insights to the field of autonomous driving but also serves as a guiding reference for developing more robust 3D object detection models. Besides, research work [27] reveals the vulnerability of common image classifiers to the generated adversarial viewpoints. The author implemented 'ViewFool: Adversarial Viewpoint Misclassification' which provides valuable insights into the necessity of evaluating model robustness under viewpoint changes. By introducing a novel method and a dedicated dataset, the study offers a foundation for enhancing viewpoint robustness and highlights the broader challenge of distribution shifts in visual recognition models.

The use of edge computing in the Rocket system for video analytics was discussed in [28]. Ganesh et al. evaluated the performance requirements of video-analytic systems used in smart cities, self-driving cars, and smart vehicles. The research concludes that edge computing is a viable approach for meeting the real-time requirements of live video analytics. Edge computing brings computation and data storage closer to the data sources. So, they reduce latency and increase the effectiveness of data processing and analysis. They concluded that edge computing is appropriate for video-analytic systems that require quick response times and real-time processing.

Ali et al. [29] developed a scalable and efficient edge-enhanced video stream processing system to process video streams in real-time. They deployed all stages on edge and utilizing in-transit resources. By doing so, the proposed system achieves real-time performance of 100 frames per second. In contrast, the cloud-only approach achieves only 15 frames per second, highlighting the significant performance improvement offered by the proposed system. This developed model has the potential to improve the efficiency and effectiveness of vision data analysis activities significantly.

Besides container runtime, container orchestration tools are also used in Edge computing. Research work [30] explored container orchestration performances. They implemented the Phoronix test suite benchmarks to evaluate the performance of Docker Swarm and Kubernetes. They put the

processor through its paces with compression, audio encoding, video encoding, and SQLite. Memory performance was measured using synthetic benchmark RAMspeed, STREAM, and CacheBench. They assess the disk performance using AIO and IOzone. The final outcome of the research was Docker Swarm is more lightweight and Kubernetes has higher overhead than Docker Swarm.

Liu et al. [31] evaluates the performance of containerization in edge-cloud computing stacks for industrial applications from a client perspective. The authors introduce a methodology for assessing the latency and processing capability of machine learning tasks across the entire stack. To validate their approach, experiments are conducted using Microsoft Azure IoT Edge and a Raspberry Pi 3B+ board. The results show that while containerization offers flexibility and scalability, it does not necessarily improve latency performance compared to cloud deployment. The study concludes that the current edge-cloud infrastructure needs to improve to fulfill industry automation needs and suggests that a static partitioning strategy can be utilized for industrial applications.

Benchmarking container orchestration was done in [32]. They proposed a benchmarking framework called COFFEE for container orchestration frameworks, which includes seven core requirements and associated performance metrics. The approach is exemplified through case studies of the Kubernetes and Nomad frameworks, both in a self-hosted environment and on the Google Cloud Platform. The author concluded that COFFEE can capture several performance metrics for container orchestration frameworks and that Kubernetes outperforms Nomad in many scenarios.

Assessing the performance of container technologies represents a compelling avenue of research. Investigations into performance evaluations among containers and virtual machines have been undertaken [33]. Additionally, studies have delved into the comparison of container performance by employing synthetic tools to scrutinize CPU, RAM, and network efficiencies [34], [35]. The findings of these inquiries consistently underscore Docker's supremacy among containers, outperforming virtual machines as well.

Table 1 provides an overview of prior research efforts in the domain of container technology within edge computing. The majority of these studies have focused on the deployment of different container runtimes or container orchestrators within edge computing environments. However, it is noteworthy that only a limited number of these investigations have specifically utilized ARM architecture for container deployment. Additionally, many of these research endeavors have employed synthetic benchmarking tools such as Sysbench, Sockperf, Ramspeed, and Tiobench to assess the performance of container technology.

Prior research on container technology benchmarking primarily relies on synthetic benchmarking tools, which execute predefined programs inside containers. For instance, Sysbench is commonly used for calculating prime numbers. In contrast, our study introduces a novel approach by designing specific applications tailored for benchmarking computer vision and data science tasks within containers. Moreover, none of the previous research addresses container efficiency for Edge computing. So, in order to find a suitable container for edge computing, we benchmark the popular container technologies. We quantitatively compare the performance of different container technologies, going beyond qualitative assessments. This numeric comparison enables developers and practitioners to make informed decisions based on specific performance metrics. To the best of our knowledge, this is the first work in the field of container benchmarking that focuses on evaluating performance for computer vision and data science applications.

III. BACKGROUND

The section focuses on the term Container technology and Edge Container. We explain the technical term container runtime and the different containers used in our experiment.

A. CONTAINER TECHNOLOGY

Containers are operating system-level virtualization that provides an isolated environment on a single host rather than utilizing a distinct operating system like virtual machines. Because of the Namespace and Cgroup kernel mechanisms, this isolation is achievable. Container runtimes, on the other hand, execute and manage the components required to run a container. As a result, container runtimes are an essential component of container management, and they often make secure execution and efficient container deployment easier [50]. In summary, a container is a packed application with dependencies, whereas a container runtime is the software that runs and manages containers on a specific system. In our experiment, we use Docker, Podman, and Singularity containers.

Docker stands out as the most extensively adopted container technology that is directly managed by the host kernel. Docker containers are configured through a *Dockerfile*, which comprises command-line interface (CLI) instructions and initial tasks. Docker images are generated using these Dockerfiles, encompassing all the executable source code, libraries, and dependencies required to instantiate Docker containers. These images are immutable and consist of multiple layers, with new layers added at the top whenever modifications are introduced using specific Docker commands. Images can be created from scratch or downloaded from Docker Hub [51].

Podman [52] is an OCI-compliant, daemonless container engine for developing, managing, and running OCI (Open Container Initiative) containers and container images. Podman follows the standards set by the OCI, enabling direct integration with the kernel, containers, image registry, and images. One notable feature of Podman is its ability to run containers as either root or rootless. Due to its robust isolation capabilities and user privilege management, Podman offers enhanced security compared to other container technologies. When running a container, Podman further enhances security

TABLE 1. Related work implemented Container technology for edge Computing.

Existing Work	Container Technology	Edge Node Hardware	ARM Architecture	Benchmark Used
Jing et al. [21]	Docker, Containered	Raspberry Pi 4	Yes	Sockperf, Ramspeed
Morabito et al. [22]	Docker	Raspberry Pi 2 and 3	Yes	MySQL, Apache2
Krivic et al. [36]	Docker	Not specified	Not specified	No
Cilic et al. [37]	LXC	VM: 2 vCPU, 2 GB RAM	No	No
Wong et al. [38]	Docker	VM: 1 vCPU, 1 GB RAM	No	No
Cicconetti et al. [39]	Apache OpenWhisk	Intel Xeon CPU	No	No
Taherizadeh et al. [40]	Linux container	Raspberry Pi 3	Yes	No
Muhammad et al. [41]	Docker	Raspberry Pi 3	Yes	LINPACK, STREAM
Pahl et al. [42]	Docker	Not Specified	Not specified	No
Ermolenko et al. [43]	Kubernetes	VM: 4 vCPU, 3GB RAM	No	No
Orive et al. [44]	K3s	Not specified	Not specified	No
Yang et al. [45]	Kubernetes, KubeEdge	ARM64 4 core CPU, 4 GB RAM	Yes	No
Lan et al. [46]	KubeEdge	Jetson AGX Xavier; Jetson nano, Raspberry Pi 3	Yes	3-D SLAM application
Fernandez et al. [47]	Kubernetes	Raspberry Pi 3	Yes	No
Kim et al. [48]	Docker, Kubernetes	Intel Xeon E5-2609	No	No
Yin et al. [49]	OpenStack	Intel Core i7-6700 CPU	No	No

by implementing an additional isolation layer through UID separation mechanisms utilizing namespaces.

Singularity [53] is a container technology designed primarily for High-Performance Computing (HPC). It is based on Linux containers, and in a single configuration file, it integrates multiple software stacks. This single file(.sif) can create and distribute containers across several platforms. Singularity solves some of the critical concerns of containers, such as compute mobility, reproducibility, HPC support, and security. The Singularity runtime supports integration and separation, making it easy to read and write data to the host system while also taking advantage of high-speed interconnects and GPUs. Furthermore, Singularity Registry HPC (shpc) facilitates the portability and reproducibility of complicated software stacks.

B. EDGE CONTAINER

Edge containers refer to decentralized containerized applications or workloads deployed at the edge of a network. Edge containers are located as close as the data source or end-user devices. It leverages container technology to enable efficient and scalable deployment of applications in edge computing environments. The difference between cloud containers and edge containers is location. In comparison, cloud containers run in distant continental or regional data centers, and edge containers run at the network's edge, significantly closer to the end user. Edge containers have become a key tool for organizations looking to decentralize their services. By moving key application components to the edge of the network, organizations can trim their network costs while also improving response times. This shift in intelligence deployment is proving to be a powerful strategy for modern organizations seeking to optimize their operations. Edgegap, a revolutionary gaming firm, unveiled a gaming platform in 2019 that reduced latency by 58% by utilizing real-time telemetry and edge containers.

IV. BENCHMARKING APPROACH

In our experiment, we implement a container-based edge system where IoT devices send data to the edge nodes. We use Raspberry Pi 4 as an edge node. An overview of our proposed system is shown in Figure 1. We meticulously designed benchmarking schemes to evaluate the performance of various container technologies on ARM-based edge devices. These schemes encompassed a series of systematic steps and metrics to ensure a comprehensive assessment. The components and functionality of our benchmark are shown in 2.

The benchmarking process commences by receiving data from IoT devices and subsequently processing it within the container deployed as an edge node. Comprehensive details of our benchmark are outlined below.

- Containers are inside the edge nodes. We used three container technologies, i.e., Docker, Podman, and Singularity. The container has all the necessary libraries and packages. The edge node receives images and Fitbit fitness tracker data from IoT devices, performs image and data analysis, and conducts necessary actions. Then the analyzed image and data are sent to the cloud for further processing.
- To benchmark the containers, we have two connection types: wired and wireless connection. We run the experiment for wire-to-wire and wireless-to-wireless. Wired and wireless connections connect both IoT devices and edge nodes.
- We have two sets of applications: Computer vision and Data Science. Every second, the container receives an image or chunk of data from IoT sensors. In case I, the images are sent by IoT sensors to the edge node along with their name and size for processing. When the container receives the image, it does the detection process, saves the output image within the container, and finally saves time and resource usage in the file. The container saves the output image as per the detected name, e.g., Face, Vehicle, Body, or Object, and with a JPG extension. Regarding fitness data, we set specific tasks for data analysis. The container receives all the data, saves it inside, conducts data cleaning, performs analysis of the data, and transmits the desired data to the cloud.

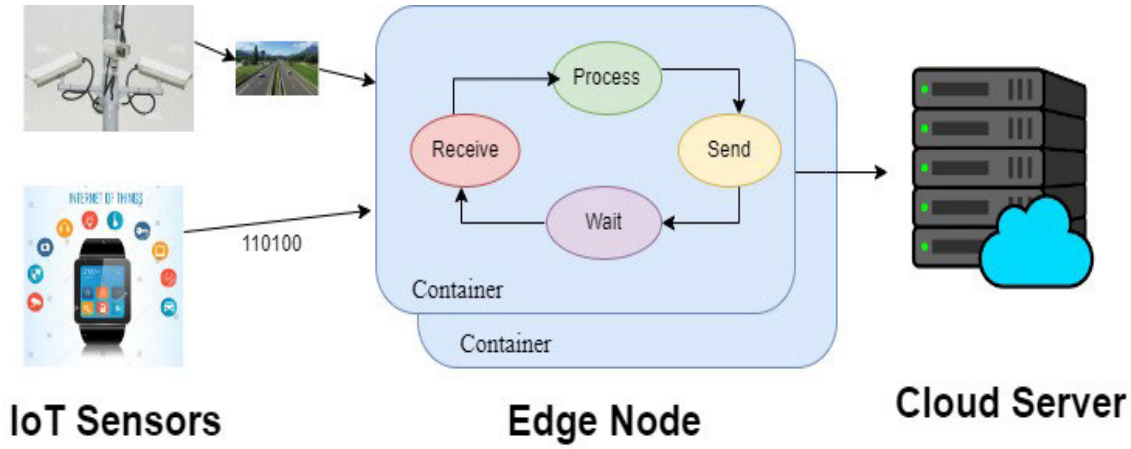

FIGURE 1. Overview of proposed System.

- The metrics to benchmark containers are computing power and network performance. By employing these metrics, we comprehensively evaluate the performance and efficiency of different container technologies in various scenarios, encompassing computer vision and data science workloads. This multifaceted approach ensures a holistic assessment of container technologies' capabilities and limitations, providing valuable insights for selecting the most appropriate technology for specific edge computing use cases.

1) Waiting Time: Measures the time an application spends in a queue before it starts executing. It provides insights into the efficiency of container scheduling and resource allocation.
2) Receiving Time: This metric gauges the time taken by containers to receive data or images from IoT devices. It reflects the responsiveness of containers to incoming data streams.
3) Processing Time: The processing time metric quantifies the duration taken by containers to analyze and process data or images. It indicates the efficiency of the underlying algorithms and computational capabilities.
4) Resource Utilization: This metric assesses how effectively containers utilize system resources, including CPU and RAM. It provides insights into the efficiency of resource management within containers.
5) Throughput: Throughput measures the rate at which a container can process a certain number of tasks or requests. It gives an indication of the container's overall processing capacity.

V. BENCHMARK DESIGN

This section introduces the benchmarking framework for container technology.

A. EXPERIMENTAL SETUP

Our experiment deploys classic container runtime on the Raspberry Pi 4 as edge nodes, a single-board computer based on the low-power, low-cost ARM processor architecture. This looks and feels like Ubuntu on a PC, but users can encounter an entirely new approach to architecture and devices behind the scenes. We utilized the most recent Raspberry Pi 4 model. Table 2 gives an overview of the specification of our experimental setup.

TABLE 2. Specification of experimental setup.

Description	Specification
Architecture	ARM 64
SoC	Broadcom BCM2711
Processor	ARM 1.5GHz CPU
Total Cores	Quad core Cortex A-72
Total RAM	4GB
OS	Linux
Power Consumption	500 mA
Micro-SD card	Available
Wireless	2.4 GHz and 5GHz
Bluetooth	5.0

We use Ubuntu 22.04, a 64-bit OS, on the Raspberry Pi. Besides, we use an ASUS RT-N56U Dual Band Wireless N600 Gigabit router for wired and wireless configuration to conduct the network experiments. It features 802.11 N wireless technology and 10/100/1000 Mbps LAN ports. We utilize *psutil* python system and process utilization library and *docker stats, podman stats* command to monitor the resources consumed by each running container.

B. CONTAINER BUILDUP

In our experiment, we created three different types of containers using Ubuntu 22.04 operating system. For the Docker container, we wrote the necessary packages and dependencies in *Dockerfile*. We downloaded the OpenCV library needed to run the computer vision application. For the Podman and Singularity containers, we build the container using the Dockerfile to maintain similar installation and dependencies in both containers. After building the container, we remove all the unnecessary packages and temporary files to receive optimized containers for our experiment.

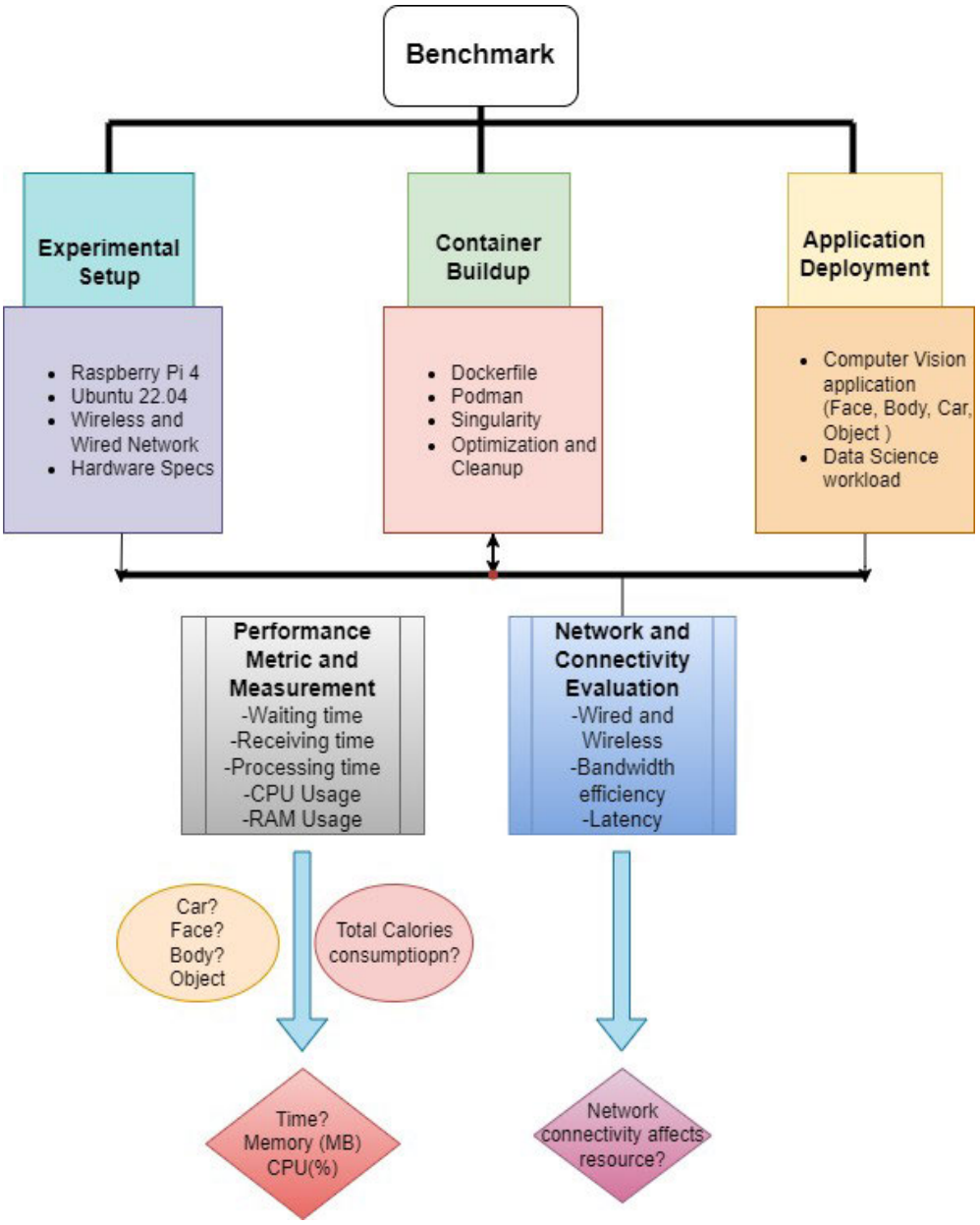

FIGURE 2. Benchmark components dataflow.

C. APPLICATION USED FOR BENCHMARKING

To benchmark container performances, we deploy four computer vision applications, i.e., Face detection, Body detection, Vehicle detection, and Object detection The computer vision applications use different algorithms i.e., Haar Cascades, Histogram of Oriented Gradients (HOG), and Convolutional Neural Networks (CNN) with YOLO algorithm to identify the image. These algorithms are often used in edge computing for object detection. In addition, they provide excellent real-time object detection performance, enabling the fast and efficient processing of video streams or live camera feeds. As IoT sensors send real-time data to edge devices, these algorithms are a perfect fit for our scenario. All the algorithm implementation and sender scripts are available in GitHub [20].

D. BENCHMARK CRITERIA

For each application, we collected various metrics, including waiting time, receiving time, processing time, CPU usage, and RAM usage. We consider the CPU usage of the container as percentages, RAM usage as MB, the total number of bytes received and transmitted over the network as NET I/O, Total number of bytes written and read from the container file system as BLOCK I/O. These metrics were recorded by the containers, and statistical analyses such as average, mean, median, maximum, minimum, and standard deviation

were calculated to benchmark their performance. To maintain uniformity, we utilized the same OpenCV library, sender, and receiver scripts across all containers and native systems, written in Python.

In the computer vision application experiments, each application was tested within each type of container using a dataset consisting of 1000 images with dimensions of 1280 × 720 pixels. Two distinct sets of images were employed: the face and body detection applications shared one set, while the vehicle detection application utilized a separate set of 1000 images. The object detection application combined 500 images from the first set (face and body) with 500 images from the second set (vehicle) to form its dataset. i For the data science workload, we use Fitbit fitness tracker data [54]. We only work with Daily Activity datasets to maintain consistency and standard parameters for each container. It has some attributes, e.g., *ActivityDate, TotalSteps, TotalDistance, Calories* etc. We send the data to each of the containers. The container receives the data, cleans, processes, and analyzes it. The final expected outcome was to calculate the average number of total steps for each user and find the user with a maximum number of average steps. Each container saves processing time, CPU, and RAM usage for performing the task.

E. NETWORKING CONFIGURATION

We examined two types of network connections: wired and wireless, to assess the impact of network conditions on container performance. Realistic network scenarios were emulated to simulate practical edge computing environments.

F. DATA COLLECTION

The containers were configured to collect data on each of the benchmarking metrics during the execution of applications. This data was recorded and analyzed for subsequent evaluation. We obtain the images from two video clips for our experiment [55], [56]. In addition, we deploy data science workload to benchmark containers for stream data. We simulated stream-like behavior by processing the Fitbit data in smaller chunks or batches, mimicking the processing of real-time or streaming data.

By meticulously implementing these benchmarking schemes, we were able to obtain comprehensive insights into the strengths and limitations of each container technology in the context of ARM-based edge devices. The resulting data not only informed our conclusions but also offered practical guidance for selecting the most appropriate container technology for specific edge computing use cases.

VI. EXPERIMENTAL RESULTS

In this section, we discussed the outcome of the experiment. Each container or system executed one application at a time, ensuring consistency and standardized comparison. By comparing the performance of different containers, we aimed to assess their suitability and effectiveness for edge computing scenarios.

A. DATA SCIENCE WORKLOAD

For the Data Science workload, data is sent in chunks or batches every 5 seconds. We observed consistent waiting times for both containers and native systems. Among the containers, Singularity demonstrated excellent performance, with processing and receiving times almost matching native systems. Interestingly, Singularity even outperformed the native system in terms of processing time for the given task. Regarding CPU usage, Docker containers exhibited the highest consumption (0.28%), while Singularity containers showed the lowest CPU usage (0.25%), comparable to the native system. However, Podman containers consumed the most memory (72.51MB) compared to all other containers, whereas the native system utilized only 50.91MB. The comprehensive results for the data science workload are presented in Figure 3.

The findings indicate that Singularity containers are well-suited for data science workloads, delivering near-native performance and efficient resource utilization. Singularity's superior performance compared to other containers in the context of the data science workload can be attributed to several key factors. Singularity containers are designed to run with minimal overhead. They are optimized for high-performance computing (HPC) environments and scientific workloads, where performance and efficiency are critical. Unlike traditional container runtimes like Docker, Singularity does not require a separate daemon to manage containers, reducing unnecessary overhead.

B. COMPUTER VISION APPLICATION

In the wireless-wireless experiment, we observed varying CPU usage for each application across different containers. For the Vehicle detection application, the containers exhibited an average CPU usage ranging from 26% to 27%, while the native system showed approximately 25% CPU usage. Notably, the Docker container had an average CPU usage of 51.44% for Face detection, the lowest CPU usage of all containers. Figure 4 provides an overview of the CPU usage across different containers for all applications. Regarding Body and Object detection, both Podman and Singularity containers demonstrated slightly better CPU utilization compared to Docker. It is worth mentioning that for these applications, the average CPU usage nearly doubled compared to face and vehicle detection. This increase in CPU usage can be attributed to the implementation of deep neural network algorithms, which require substantial computational resources for processing and classifying various object classes.

We also examined the RAM usage patterns across different containers for each application. Figure 5 illustrates the trends in RAM utilization. For both the Face and Car detection applications, we observed similar ranges of RAM

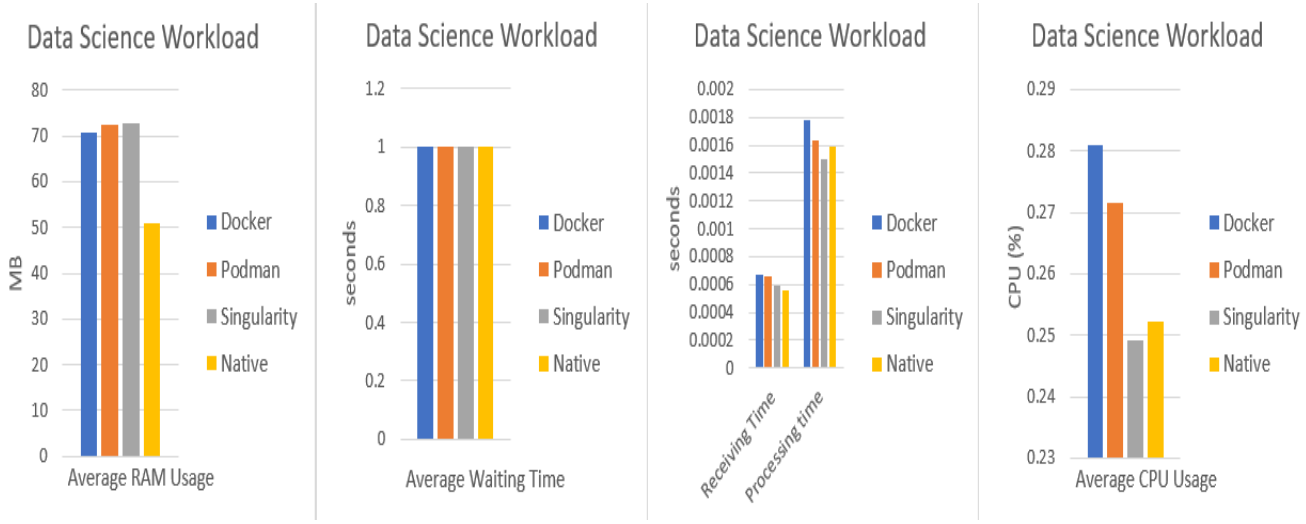

FIGURE 3. Average receiving, processing, waiting time and average CPU and RAM usage for different containers for data science application.

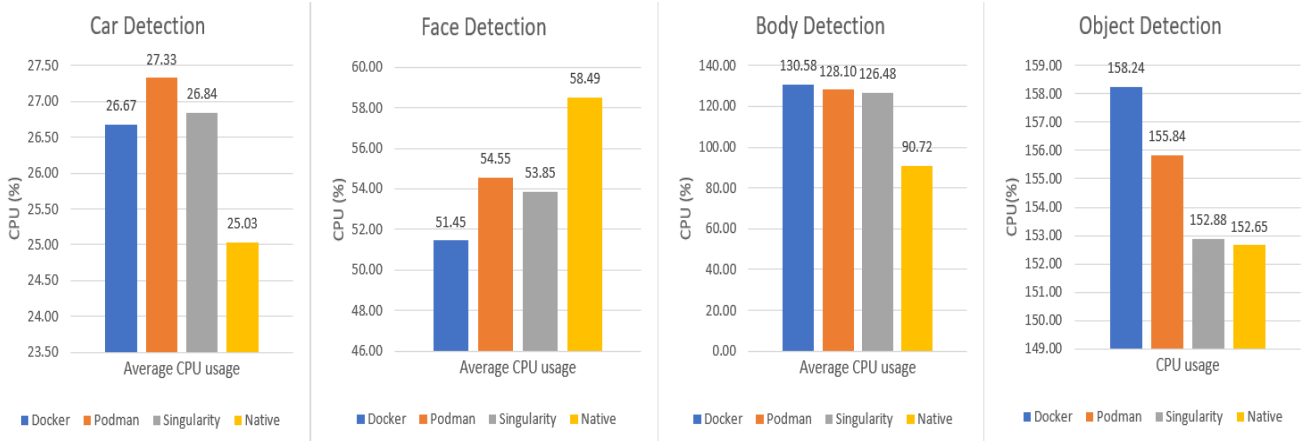

FIGURE 4. Average CPU usage for different containers for computer vision application in wireless-wireless connection.

usage, ranging from 90-96 MB for the containers and approximately 79 MB for the native systems. The Body detection application exhibited relatively lower RAM usage, averaging around 80-84 MB. However, the RAM usage significantly increased for the object detection application, which can be attributed to the utilization of deep neural network algorithms.

We further benchmark container performance based on receiving, processing, and waiting time. They are critical in edge computing because they affect real-time responsiveness, low latency, and bandwidth efficiency. Figure 6 illustrates the overall scenario for computer vision applications. In the case of receiving images from IoT sensors, containers have shown better performance than native systems in Car, Face, and Body detection. However, in the case of object detection applications, the receiving time is the same for both the podman container and the native system. The singularity container has the slowest receiving rate among the containers. The average receiving time ranges between.09-.15 seconds for all computer vision applications.

Regarding waiting time, containers perform similarly for Car, Face, and Body detection applications. However, it varies in Object detection applications. Docker container has the lowest waiting time among containers, with around .9 seconds, giving a similar performance as the native system.

The average processing or execution time for an application defines how much time it is needed for containers to identify an image correctly. The average processing time for Car detection, Face detection, and Body detection is .12, .2, and .4 seconds, respectively. For object detection, Docker spent the most time among containers for processing, with an average of 1.3 seconds; however, Singularity and Podman performed better in this case. The native systems have shown the most improvement in detecting an Object with an average of .98 seconds. Overall, the Car detection application has the fastest processing time for all containers,

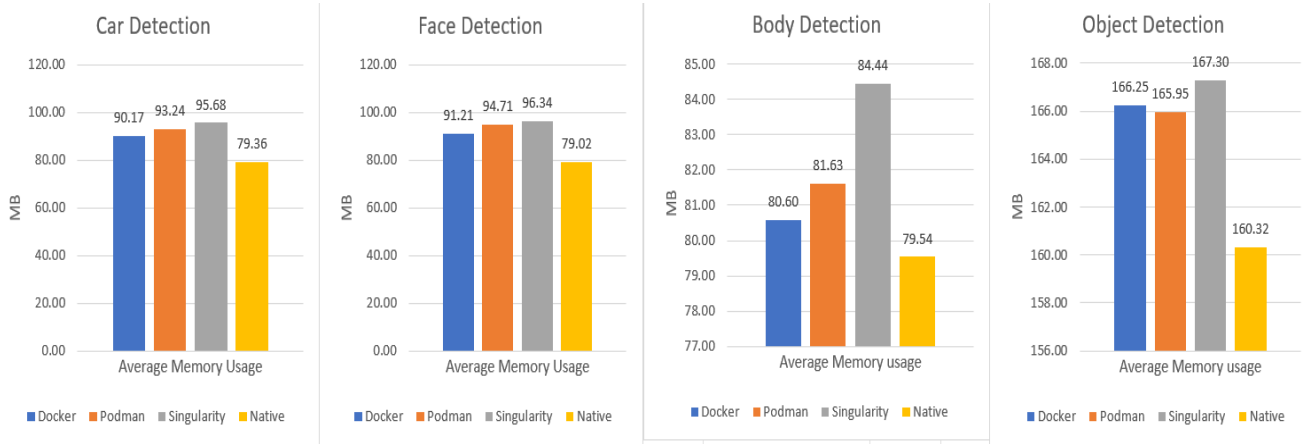

FIGURE 5. Average RAM usage for different containers for computer vision application in wireless-wireless connection.

FIGURE 6. Average receiving, processing, and waiting time for different containers for computer vision application in wireless-wireless connection.

followed by Face detection and Body detection. The Object detection application shows the slowest processing time. The experiment highlights the processing time as a key performance metric when evaluating the effectiveness of the Object detection algorithm. The extended execution time observed in the Object detection application deployed on edge devices can be attributed to the resource-intensive nature of deep neural network algorithms utilized. These algorithms demand significant computational power and memory resources to classify a wide range of object classes effectively. Particularly, the Object detection algorithm employed in this study encompasses the identification and classification of over 20 distinct object classes, necessitating the utilization of a Graphics Processing Unit (GPU) for optimal performance. Consequently, when relying solely on the Central Processing Unit (CPU) for executing the object detection application, the available RAM becomes heavily utilized, resulting in prolonged execution times.

Our findings show that network connection does not have a notable impact on CPU and RAM for containers but for native systems. The containers have consistency in CPU and RAM consumption in wired-wired networks. On the other hand, The native system in a wired-wired connection has lower RAM consumption and higher CPU consumption than wireless-wireless. Figure 7 and Figure 8 show the trend of CPU and RAM usage for computer vision applications.

For Car detection applications, the average CPU usage for containers is 26% and 32.93% for native systems. For Face detection, the CPU usage ranges between 54-55.6% for containers and 62.15% for the native. The containers have almost similar CPU usage for Body detection applications, ranging between 128.55-129.44%, and the native system has 155.48%. The containers exhibit near-native performances in these three applications; however, for Object detection, the Podman container has the lowest CPU usage. One possible explanation is that there might be some background

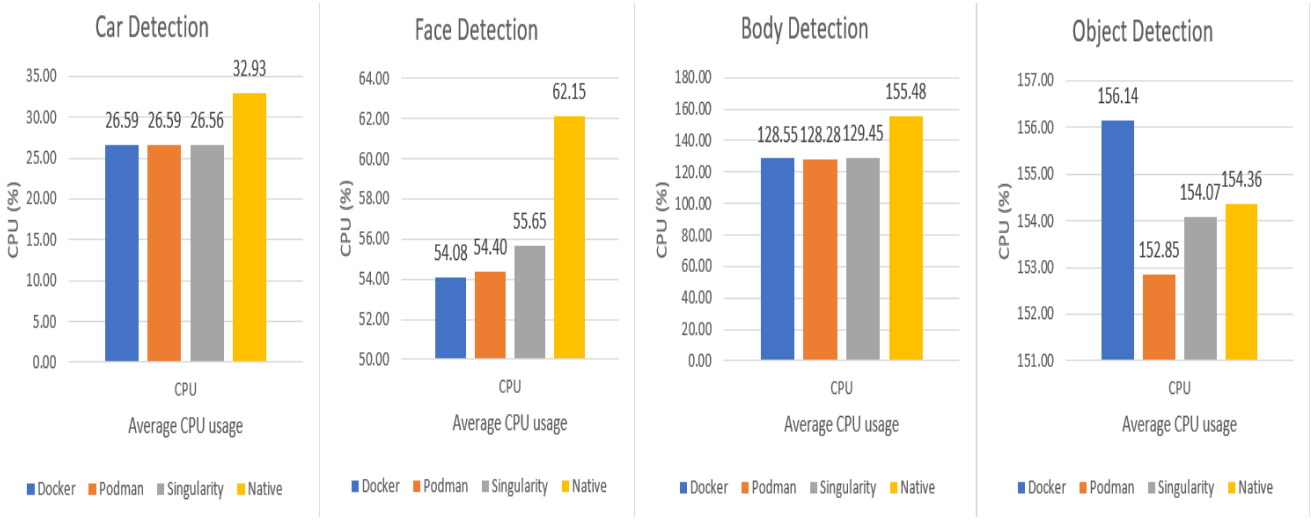

FIGURE 7. Average CPU usage for different containers for computer vision application in wired-wired connection.

FIGURE 8. Average memory usage for different containers for computer vision application in wired-wired connection.

tasks that stress native systems CPU. However, containers are isolated from the host, so those background tasks can not stress the container CPU. All the containers have more RAM consumption than the native system for all four applications. The containers are consistent in terms of RAM usage.

Our findings reveal that waiting and processing times for both containers and native systems are consistent for wired-wired connections. Figure 9 shows that containers achieve near-native performance for all computer vision applications regarding waiting and processing time. However, the receiving time varies between containers and native systems. We found the native system has a performance overhead regarding receiving time. For Car, Face, and Body detection applications, the receiving time is more than .1 seconds for the native system, whereas it is around .06 seconds for containers. As receiving time is a crucial criterion in edge computing, this can be a determining factor for choosing containers and communication types.

Our benchmark results are summarized in Table 3. Docker exhibits superior performance across most applications, with Singularity outperforming others in the context of data science workloads. Furthermore, we compare our findings with related research on container performance, presented in 4. Consistently, previous studies align with our results, highlighting container technology is suitable for edge computing and Docker has a performance advantage over other containers. Prior research has predominantly overlooked network latency, primarily due to limitations in synthetic benchmarks that cannot simulate data transmission to various devices. Consequently, the transmission of actual data from IoT devices to edge devices has been a neglected

FIGURE 9. Average receiving, processing, and waiting time for different containers for computer vision application in wired-wired connection.

aspect. In this study, we aim to fill this void by addressing this critical research gap.

VII. DISCUSSIONS

This section focuses on the critical aspects of the benchmark results, covering network connectivity, receiving, processing and waiting time, Memory and CPU utilization, and Operating systems. These factors are essential for achieving optimal performance and benchmarking computer vision and data science applications deployed within containers.

A. CONTAINER SELECTION

In the realm of ARM architecture, almost all popular container technologies are supported, although some may lack certain features specific to ARM. Among these technologies, Docker stands out as the most popular and feature-rich choice. It offers rapid deployment, a small footprint, and excellent performance. Docker's versatility in shifting between different locations makes it particularly well-suited for edge computing and IoT scenarios. Additionally, Docker Hub provides extensive support for pre-built images designed for computer vision and data science applications. With its vast ecosystem of tools and libraries, Docker emerges as the most suitable and widely adopted container technology.

Podman presents a similar user experience to Docker but with the added advantage of being daemonless and providing enhanced security. Recent advancements in Podman include automating the management of unhealthy containers, which is crucial in remote and critical systems. As edge computing demands self-robustness, self-healing capabilities, and automated deployments, Podman becomes an appealing choice for edge computing scenarios.

Singularity containers operate within the user space, ensuring identical user permissions inside and outside the container. Singularity is well-suited for high-performance computing (HPC) and computer vision applications involving large-scale batch-scheduling jobs. It seamlessly integrates with HPC clouds, offering near-native performance and leveraging cloud features such as simplified management and deployment.

We also started the experiment with the LXC container. While experimenting, we found issues with data receiving from IoT devices. Moreover, LXC will discontinue its support in 2027. So, we did not include it in our benchmarking.

B. NETWORK CONNECTION

The type of network connection, whether wireless-wireless or wired-wired, can have a significant impact on edge computing depending on the workload and application type. Each connection type has its own characteristics that can influence the performance and reliability of containers or native systems in an IoT-Edge scenario. If the connection is stopped every time after sending an image or a chunk of data, the system will be optimized and speed up the transmission time. However, it can also slower overall transmission time and may generate congestion. Further experiments are necessary to evaluate the impact of the optimization, considering factors like container runtime, network types, image sizes, and processing speeds. Careful assessment of potential drawbacks is vital before implementing the approach in real systems.

C. CPU USAGE

The analysis of CPU usage has provided valuable insights into the performance of container technologies in the context of edge computing on ARM-based devices. The results demonstrate variations in CPU utilization across different container types and applications. We found CPU utilization was consistent for all three containers in the wired-wired version. However, in wireless Singularity container has the best CPU utilization among the containers. With more complex applications like Object detection applications which consume more CPU, Singularity excels among the containers. The observed higher CPU usage in Object detection applications, driven by deep neural network

TABLE 3. Benchmarks for container runtimes for computer vision and data science application on ARM-based edge device.

Application	Criteria	Performance Metrics	Docker	Podman	Singularity	Native	Best
	lower is better	Waiting Time(sec)	.92	.91	.90	.95	Singularity
	lower is better	Receiving Time(sec)	.10	.09	.08	.15	Singularity
Car Detection	lower is better	Processing Time(sec)	.12	.12	.13	.09	Docker
	lower is better	CPU usage(%)	26.67	27.33	26.84	25.03	Docker
	lower is better	RAM usage(MB)	90.17	93.24	95.68	79.36	Docker
	lower is better	Waiting Time(sec)	.83	.83	.83	.84	Tie
	lower is better	Receiving Time(sec)	.09	.10	.16	.13	Docker
Face Detection	lower is better	Processing Time(sec)	.21	.20	.20	.19	Tie
	lower is better	CPU usage(%)	51.45	54.55	53.85	58.49	Docker
	lower is better	RAM usage(MB)	91.21	94.71	96.34	79.02	Docker
	lower is better	Waiting Time(sec)	.63	.64	.64	.76	Docker
	lower is better	Receiving Time(sec)	.08	.09	.12	.13	Docker
Body Detection	lower is better	Processing Time(sec)	.40	.39	.40	.28	Podman
	lower is better	CPU usage(%)	130.58	128.1	126.48	90.72	Singularity
	lower is better	RAM usage(MB)	80.59	81.63	84.44	79.54	Docker
	lower is better	Waiting Time(sec)	.86	1.01	1.05	.85	Docker
	lower is better	Receiving Time(sec)	.13	.09	.14	.09	Podman
Object Detection	lower is better	Processing Time(sec)	1.27	1.03	.99	.98	Singularity
	lower is better	CPU usage(%)	160.24	155.84	148.88	143.65	Singularity
	lower is better	RAM usage(MB)	166.25	165.95	167.30	160.32	Podman
	lower is better	Waiting Time(sec)	.99	.98	.99	.99	Tie
	lower is better	Receiving Time(sec)	.001	.001	.001	.001	Tie
Data Science Workload	lower is better	Processing Time(sec)	.002	.002	.001	.001	Singularity
	lower is better	CPU usage(%)	.28	.27	.24	.25	Singularity
	lower is better	RAM usage(MB)	70.84	72.51	72.74	50.91	Docker

TABLE 4. Performance comparison of container technologies and baseline solutions.

Existing Work	Container Evaluated	Performance Metric	Conclusion
Morabito et al. [23]	Docker, LXC vs KVM	CPU, Disk I/o, Memory, Network I/o	Container performs better
Espe et al. [35]	containerd, crio	CPU, Memory, Disk I/O	containerd perform better
Avino et al. [57]	Docker	CPU	CPU overhead independent of Clients number
Casalicchio et al. [58]	Docker	CPU, Disk I/O	Need more research
Kozhirbayev et al. [59]	Docker, LXC	CPU, Memory, Disk I/O, Network	CPU:Docker; Disk:LXC; Memory:Docker; Network:Docker
Xavier et al. [60]	LXC, Linux-server, OpenVZ	CPU, Memory, Disk, HPC	LXC better suited for HPC than VM
Chung et al. [61]	Docker	HPC	Docker has less overhead in HPC
Lee et al. [62]	Docker	CPU, Network, I/O	Container is suitable in server-less computing
Our Proposed Benchmark	*Docker, Podman, Singularity*	*Waiting, Receiving, Processing Time CPU, Memory, Network performance*	*Docker is the most suitable for Edge computing*

algorithms, raises the importance of optimizing algorithms and container technology for improved CPU efficiency in edge environments.

Incorporating a GPU has the potential to significantly enhance processing speed and efficiency, which could effectively address any concerns related to CPU usage. To delve deeper into this prospect, conducting comprehensive tests and comparisons involving the GPU would be advantageous. This could entail running identical tasks with and without the GPU to discern potential disparities in CPU utilization. Such investigations would provide valuable insights into the impact of GPU acceleration on containerized edge applications, guiding the optimization of resource utilization and overall performance in edge computing environments.

D. MEMORY USAGE

Our findings underscore the importance of considering memory usage as a key factor in selecting container technologies for edge computing on ARM-based devices. We found memory usage has not posed any significant bottlenecks in the current system, with no reported issues. The majority of our applications exhibit low memory

consumption, typically utilizing less than 3% of the available memory. However, the object detection application stands out as an exception, consuming approximately 5-6% of the memory.

Among the container runtimes evaluated, Docker, Podman, and Singularity exhibited varying patterns of RAM utilization. Docker showcased efficient memory management, maintaining consistent and relatively low RAM usage for all tested applications. Podman, as a daemonless alternative to Docker, also demonstrated commendable RAM efficiency, making it a suitable option for resource-constrained edge devices. On the other hand, Singularity exhibited slightly higher RAM usage for certain applications.

Optimizing memory usage in edge computing is crucial due to the limited resources available on edge devices. Containers with efficient memory management help maximize resource utilization and enhance the scalability of edge applications. Minimizing RAM usage is essential to avoid potential bottlenecks and ensure the smooth operation of multiple containers on edge devices. Container image size and content play a significant role in determining RAM usage. Adopting smaller and more optimized container images, removing unnecessary dependencies, and using lightweight base images can help reduce memory overhead.

E. RECEIVING TIME

The performance evaluation of different container technologies (Docker, Podman, Singularity) is crucial to determine their suitability for processing vast amounts of data generated by IoT devices. The efficiency of container technologies in handling data reception can significantly impact the real-time processing capabilities of edge devices.

It is essential to highlight that for containers, wired connections do not exhibit notable issues concerning receiving time. This observation is likely attributed to the inherent advantages of wired connections, which provide a stable and reliable data transmission environment unaffected by external factors like wireless signal interference. As a result, the receiving time for wired versions remains consistent and dependable for containers, contributing to the overall efficiency of edge computing systems.

The results of our analysis demonstrate variations in receiving time in wireless-wireless connection. Docker exhibited efficient(lowest) receiving times for most applications. Podman also demonstrated promising receiving times(closer to Docker), making it a viable option for edge computing environments. Singularity containers, known for their suitability in HPC and large-scale batch scheduling jobs, showed overall higher receiving times in wired-wired connections for computer vision applications but lowest for Data Science workload.

F. PROCESSING TIME

In the context of processing time, our experimental results revealed significant variations among the different container technologies. Docker exhibited competitive processing times across various computer vision and data science applications, making it a suitable choice for resource-constrained edge environments. Podman showcased efficient processing capabilities, particularly beneficial for critical systems and remote locations. However, it slightly trailed behind Docker in certain scenarios. Singularity, designed for high-performance computing and large-scale batch scheduling jobs, demonstrated remarkable processing capabilities for computer vision applications that involve extensive algorithmic computations. The variations in processing time can be attributed to the unique design principles and optimizations employed by each container technology. Docker's widespread adoption and rich ecosystem have contributed to its optimized performance, while Podman's secure and self-healing features are favorable for edge computing scenarios. Singularity, tailored for HPC tasks, exhibited promising results for specific computer vision applications where performance is paramount. To further enhance processing efficiency, future research could explore container orchestration and scheduling mechanisms that intelligently allocate computing resources based on real-time demands. Additionally, optimizing containe1r images and fine-tuning runtime configurations can contribute to better performance across various application scenarios.

G. WAITING TIME

One of the essential aspects of benchmarking containers is waiting time in the IoT-Edge environment. However, waiting time depends mostly on how the experimental setup is done and on the application type. Our benchmark results indicate that Docker and Podman showcased relatively low waiting times, making them efficient choices for edge computing environments with rapid task execution demands. Docker's mature ecosystem and widespread usage have contributed to its optimized scheduling capabilities, allowing for quick task deployment and execution. Similarly, Podman's recent features for the automated handling of unhealthy containers have contributed to its reduced waiting time. Conversely, Singularity, designed primarily for high-performance computing (HPC) applications, exhibited marginally higher waiting times in certain scenarios. This outcome could be attributed to Singularity's focus on batch scheduling and its priority for larger-scale HPC tasks rather than quick task execution typical in edge computing scenarios.

H. OPERATING SYSTEM

The decision to use Ubuntu as the operating system was based on its popularity and the wide range of available packages, ensuring a comprehensive evaluation of container performance. The benchmark can be used with other OS too. This is one of the advantages of container technology, as they can easily be ported into any OS. In the future, we will experiment with the benchmark for other OS,

i.e., Apline or Debian. These alternative operating systems may offer different advantages and trade-offs, particularly concerning container size and resource usage. The benchmark can determine the best-fit OS for ARM-based edge devices.

VIII. CONCLUSION

Our benchmarking study emphasizes the suitability and effectiveness of container technologies, such as Docker, Podman, and Singularity, for deploying computer vision and data science applications on ARM-based edge devices. We benchmark the containers based on resource consumption and network types. Our experiment detects that resource usage varies for containers and applications. We found Singularity containers perform well for data science workloads, while Docker containers have lower CPU usage for face detection as computer vision applications. This suggests that different container technologies may have strengths and limitations depending on the specific workload. Besides, Network connection does not significantly affect CPU and RAM usage for containers, but it does for native systems. Containers achieve near-native performance in terms of waiting and processing time. However, the receiving time varies between containers and native systems. This indicates that containers can provide efficient and real-time processing capabilities in edge computing scenarios. Moreover, we found that containers provide real-time responsiveness, low latency, and bandwidth efficiency, making them a practical solution for handling IoT data streams and real-time processing tasks. The choice of container technology should be based on specific application requirements and resource constraints. Containers offer a valuable tool for optimizing resource utilization and enhancing overall performance in edge computing scenarios with real-time object detection and data science workloads.

The insights gained from this study can guide developers and practitioners in selecting the most appropriate container technology for their edge computing use cases, ultimately contributing to the advancement of edge computing and IoT applications on ARM-based devices. However, we acknowledge that our study has limitations. We primarily focus on ARM-based edge devices and do not explore the broader spectrum of other edge computing architectures. Future research should expand the scope to include various hardware architectures, enabling a more comprehensive understanding of container performance in diverse edge environments.

In the future, our scope will expand to benchmark more container technologies and unikernels. Additionally, we plan to include more applications such as Voice Recognition and Human Activity Recognition tasks using containerization technology, further enhancing our understanding of container performance in various edge computing scenarios. We will also add scalability to our benchmark in order to see how performance differs when the number of containers is increased.

ACKNOWLEDGMENT

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation herein.

REFERENCES

[1] X. Xu, Q. Liu, Y. Luo, K. Peng, X. Zhang, S. Meng, and L. Qi, "A computation offloading method over big data for IoT-enabled cloud-edge computing," *Future Gener. Comput. Syst.*, vol. 95, pp. 522–533, Jun. 2019.

[2] L. Tu, S. Liu, Y. Wang, C. Zhang, and P. Li, "An optimized cluster storage method for real-time big data in Internet of Things," *J. Supercomput.*, vol. 76, no. 7, pp. 5175–5191, Jul. 2020.

[3] W. Rafique, L. Qi, I. Yaqoob, M. Imran, R. U. Rasool, and W. Dou, "Complementing IoT services through software defined networking and edge computing: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 1761–1804, 3rd Quart., 2020.

[4] A. J. Ferrer, J. M. Marquès, and J. Jorba, "Towards the decentralised cloud: Survey on approaches and challenges for mobile, ad hoc, and edge computing," *ACM Comput. Surveys*, vol. 51, no. 6, pp. 1–36, Jan. 2019.

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

[6] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, May 2016.

[7] I. Sittón-Candanedo, R. S. Alonso, Ó. García, L. Muñoz, and S. Rodríguez-González, "Edge computing, IoT and social computing in smart energy scenarios," *Sensors*, vol. 19, no. 15, p. 3353, Jul. 2019.

[8] A. Martin, S. Raponi, T. Combe, and R. Di Pietro, "Docker ecosystem—Vulnerability analysis," *Comput. Commun.*, vol. 122, pp. 30–43, Jun. 2018.

[9] A. Singh, S. Garg, S. Batra, N. Kumar, and J. J. P. C. Rodrigues, "Bloom filter based optimization scheme for massive data handling in IoT environment," *Future Gener. Comput. Syst.*, vol. 82, pp. 440–449, May 2018.

[10] A. Kumari, S. Tanwar, S. Tyagi, N. Kumar, M. Maasberg, and K.-K.-R. Choo, "Multimedia big data computing and Internet of Things applications: A taxonomy and process model," *J. Netw. Comput. Appl.*, vol. 124, pp. 169–195, Dec. 2018.

[11] R. Mijat and A. Nightingale, "Virtualization is coming to a platform near you," ARM White Paper, 2011.

[12] S. A. R. Shah, A. Waqas, M.-H. Kim, T.-H. Kim, H. Yoon, and S.-Y. Noh, "Benchmarking and performance evaluations on various configurations of virtual machine and containers for cloud-based scientific workloads," *Appl. Sci.*, vol. 11, no. 3, p. 993, Jan. 2021.

[13] *Amazon-Fargate*. Accessed: Apr. 3, 2023. [Online]. Available: https://aws.amazon.com/fargate/

[14] *Kubernetes*. Accessed: May 21, 2023. [Online]. Available: https://docs.microsoft.com/en-us/learn/modules/intro-to-azure-kubernetes-service/

[15] *K3s—Lightweight Kubernetes*. Accessed: May 5, 2022. [Online]. Available: https://rancher.com

[16] Onlabs. *Tiny Titan: A Supercomputer for the Classroom*. [Online]. Available: https://tinytitan.github.io/software

[17] D. Tech. *Microsoft is Building an Online Raspberry PI Simulator*. [Online]. Available: http://developer-tech.com/news/2017/jun/21/microsoft-building-online-raspberry-pi-simulator/

[18] J. Kiepert, "Creating a raspberry pi-based Beowulf cluster," *Boise State*, 2013.

[19] C. Pahl, S. Helmer, L. Miori, J. Sanin, and B. Lee, "A container-based edge cloud PaaS architecture based on raspberry Pi clusters," in *Proc. IEEE 4th Int. Conf. Future Internet Things Cloud Workshops (FiCloudW)*, Aug. 2016, pp. 117–124.

[20] *Experimental Code*. Accessed: Jun. 13, 2023. [Online]. Available: https://github.com/dipu134420/Benchmarking-Container-Technology-on-ARM-Device

[21] Y. Jing, Z. Qiao, and R. O. Sinnott, "Benchmarking container technologies for IoT environments," in *Proc. 7th Int. Conf. Fog Mobile Edge Comput. (FMEC)*, Dec. 2022, pp. 1–8.

[22] R. Morabito, "Virtualization on Internet of Things edge devices with container technologies: A performance evaluation," *IEEE Access*, vol. 5, pp. 8835–8850, 2017.

[23] R. Morabito, "A performance evaluation of container technologies on Internet of Things devices," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2016, pp. 999–1000.

[24] R. Buyya and S. N. Srirama, *A Lightweight Container Middleware for Edge Cloud Architectures*, 2019, pp. 145–170.

[25] A. Acharya, J. Fanguède, M. Paolino, and D. Raho, "A performance benchmarking analysis of hypervisors containers and unikernels on ARMv8 and ×86 CPUs," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Jun. 2018, pp. 282–9.

[26] Y. Dong, C. Kang, J. Zhang, Z. Zhu, Y. Wang, X. Yang, H. Su, X. Wei, and J. Zhu, "Benchmarking robustness of 3D object detection to common corruptions in autonomous driving," 2023, *arXiv:2303.11040*.

[27] Y. Dong, S. Ruan, H. Su, C. Kang, X. Wei, and J. Zhu, "ViewFool: Evaluating the robustness of visual recognition to adversarial viewpoints," 2022, *arXiv:2210.03895*.

[28] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, pp. 58–67, Oct. 2017.

[29] M. Ali, A. Anjum, O. Rana, A. R. Zamani, D. Balouek-Thomert, and M. Parashar, "RES: Real-time video stream analytics using edge enhanced clouds," *IEEE Trans. Cloud Comput.*, vol. 10, no. 2, pp. 792–804, Apr. 2022.

[30] Y. Pan, I. Chen, F. Brasileiro, G. Jayaputera, and R. Sinnott, "A performance comparison of cloud-based container orchestration tools," in *Proc. IEEE Int. Conf. Big Knowl. (ICBK)*, Nov. 2019, pp. 191–198.

[31] 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, 2021.

[32] M. Straesser, J. Mathiasch, A. Bauer, and S. Kounev, "A systematic approach for benchmarking of container orchestration frameworks," in *Proc. ACM/SPEC Int. Conf. Perform. Eng.* New York, NY, USA: Association for Computing Machinery, Apr. 2023, pp. 187–198.

[33] M. Raho, A. Spyridakis, M. Paolino, and D. Raho, "KVM, Xen and Docker: A performance analysis for ARM based NFV and cloud computing," in *Proc. IEEE 3rd Workshop Adv. Inf., Electron. Electr. Eng. (AIEEE)*, Nov. 2015, pp. 1–8.

[34] Á. Kovács, "Comparison of different Linux containers," in *Proc. 40th Int. Conf. Telecommun. Signal Process. (TSP)*, Jul. 2017, pp. 47–51.

[35] L. Espe, A. Jindal, V. Podolskiy, and M. Gerndt, "Performance evaluation of container runtimes," in *Proc. 10th Int. Conf. Cloud Comput. Services Sci.*, Jan. 2020, pp. 273–281.

[36] P. Krivic, M. Kusek, I. Cavrak, and P. Skocir, "Dynamic scheduling of contextually categorised Internet of Things services in fog computing environment," *Sensors*, vol. 22, no. 2, p. 465, Jan. 2022.

[37] I. Cilic, I. P. Žarko, and M. Kušek, "Towards service orchestration for the cloud-to-thing continuum," in *Proc. 6th Int. Conf. Smart Sustain. Technol. (SpliTech)*, Sep. 2021, pp. 1–7.

[38] W. Wong, A. Zavodovski, P. Zhou, and J. Kangasharju, "Container deployment strategy for edge networking," in *Proc. 4th Workshop Middleware Edge Clouds Cloudlets (MECC)*, New York, NY, USA: Association for Computing Machinery, Dec. 2019, pp. 1–6.

[39] C. Cicconetti, M. Conti, and A. Passarella, "A decentralized framework for serverless edge computing in the Internet of Things," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 2, pp. 2166–2180, Jun. 2021.

[40] S. Taherizadeh, V. Stankovski, and M. Grobelnik, "A capillary computing architecture for dynamic Internet of Things: Orchestration of microservices from edge devices to fog and cloud providers," *Sensors*, vol. 18, no. 9, p. 2938, Sep. 2018.

[41] M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, "Orchestration of microservices for IoT using Docker and edge computing," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 118–123, Sep. 2018.

[42] C. Pahl, N. E. Ioini, S. Helmer, and B. Lee, "An architecture pattern for trusted orchestration in IoT edge clouds," in *Proc. 3rd Int. Conf. Fog Mobile Edge Comput. (FMEC)*, Apr. 2018, pp. 63–70.

[43] D. Ermolenko, C. Kilicheva, A. Muthanna, and A. Khakimov, "Internet of Things services orchestration framework based on Kubernetes and edge computing," in *Proc. IEEE Conf. Russian Young Researchers Electr. Electron. Eng. (ElConRus)*, Jan. 2021, pp. 12–17.

[44] A. Orive, A. Agirre, H.-L. Truong, I. Sarachaga, and M. Marcos, "Quality of service aware orchestration for cloud-edge continuum applications," *Sensors*, vol. 22, no. 5, p. 1755, Feb. 2022.

[45] S. Yang, Y. Ren, J. Zhang, J. Guan, and B. Li, "KubeHICE: Performance-aware container orchestration on heterogeneous-ISA architectures in cloud-edge platforms," in *Proc. IEEE Intl Conf Parallel Distrib. Process. Appl., Big Data Cloud Comput., Sustain. Comput. Commun., Social Comput. Netw. (ISPA/BDCloud/SocialCom/SustainCom)*, Sep. 2021, pp. 81–91.

[46] D. Lan, A. Taherkordi, F. Eliassen, L. Liu, S. Delbruel, S. Dustdar, and Y. Yang, "Task partitioning and orchestration on heterogeneous edge platforms: The case of vision applications," *IEEE Internet Things J.*, vol. 9, no. 10, pp. 7418–7432, May 2022.

[47] J.-M. Fernandez, I. Vidal, and F. Valera, "Enabling the orchestration of IoT slices through edge and cloud microservice platforms," *Sensors*, vol. 19, no. 13, p. 2980, Jul. 2019.

[48] S. Kim, E. Yang, and C.-H. Youn, "An accelerated edge computing with a container and its orchestration," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2019, pp. 1283–1288.

[49] L. Yin, P. Li, and J. Luo, "Smart contract service migration mechanism based on container in edge computing," *J. Parallel Distrib. Comput.*, vol. 152, pp. 157–166, Jun. 2021.

[50] S. Kaiser, M. S. Haq, A. S. Tosun, and T. Korkmaz, "Container technologies for ARM architecture: A comprehensive survey of the state-of-the-art," *IEEE Access*, vol. 10, pp. 84853–84881, 2022.

[51] Docker Hub. *Web Page*. Accessed: Oct. 3, 2021. [Online]. Available: https://hub.docker.com/

[52] Podman. *Web Page*. Accessed: Nov. 11, 2021. [Online]. Available: https://podman.io/

[53] (Nov. 2021). *SINGULARITY*. [Online]. Available: https://sylabs.io/docs/

[54] *Fitbit Fitness Tracker Data*. Accessed: Jun. 12, 2023. [Online]. Available: https://www.kaggle.com/datasets/arashnic/fitbit

[55] *Image Sample*. Accessed: May 5, 2023. [Online]. Available: https://drive.google.com/file/d/1QUClOAWRVqNJII1crbCbdStJYzq9Uhfx/view

[56] *Image Sample*. Accessed: May 5, 2023. [Online]. Available: https://www.youtube.com/watch?v=5vjn9gfHwkY

[57] G. Avino, M. Malinverno, F. Malandrino, C. Casetti, and C. F. Chiasserini, "Characterizing Docker overhead in mobile edge computing scenarios," in *Proc. Workshop Hot Topics Container Netw. Networked Syst.* New York, NY, USA: Association for Computing Machinery, Aug. 2017, pp. 30–35.

[58] E. Casalicchio and V. Perciballi, "Measuring Docker performance: What a mess!!!" in *Proc. 8th ACM/SPEC Int. Conf. Perform. Eng. Companion*. New York, NY, USA: Association for Computing Machinery, Apr. 2017, pp. 11–16.

[59] Z. Kozhirbayev and R. O. Sinnott, "A performance comparison of container-based technologies for the cloud," *Future Gener. Comput. Syst.*, vol. 68, pp. 175–182, Mar. 2017.

[60] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *Proc. 21st Euromicro Int. Conf. Parallel, Distrib., Network-Based Process.*, Feb. 2013, pp. 233–240.

[61] M. T. Chung, N. Quang-Hung, M.-T. Nguyen, and N. Thoai, "Using Docker in high performance computing applications," in *Proc. IEEE 6th Int. Conf. Commun. Electron. (ICCE)*, Jul. 2016, pp. 52–57.

[62] H. Lee, K. Satyam, and G. Fox, "Evaluation of production serverless computing environments," in *Proc. IEEE 11th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2018, pp. 442–450.

SHAHIDULLAH KAISER received the B.Sc. degree in computer science and engineering from the Islamic University of Technology (IUT), in 2017. He is currently pursuing the Ph.D. degree in computer science with The University of Texas at San Antonio. He has been a Graduate Assistant with the Department of Computer Science, since 2019. His research interests include container technology, the Internet of Things, edge computing, and privacy policy analysis.

ALI ŞAMAN TOSUN received the B.S. and M.S. degrees in computer engineering from Bilkent University, Ankara, Turkey, in 1995 and 1997, respectively, and the M.S. and Ph.D. degrees in computer science and engineering from The Ohio State University, in 1998 and 2003, respectively. From 2003 to 2021, he was an Assistant Professor with the Department of Computer Science, The University of Texas at San Antonio. He is currently the Allen C. Meadors Endowed Chair of Computer Science with The University of North Carolina at Pembroke. His research interests include network security, software-defined networks, the Internet of Things, storage systems, and large-scale data management.

TURGAY KORKMAZ received the B.Sc. degree (Hons.) in computer science and engineering from Hacettepe University, Ankara, Turkey, in 1994, the M.Sc. degree in computer engineering from Bilkent University, Ankara, in 1996, the M.Sc. degree in computer and information science from Syracuse University, Syracuse, NY, USA, in 1997, and the Ph.D. degree in electrical and computer engineering from The University of Arizona, in December 2001, under the supervision of Dr. M. Krunz. In January 2002, he joined the Department of Computer Science, The University of Texas at San Antonio, as an Assistant Professor. He is currently a Full Professor with the Department of Computer Science. He is also involved in the areas of computer networks, network security, network measurement and modeling, and internet related technologies. His research interest includes quality-of-services (QoS)-based networking issues in both wireline and wireless networks.