Storage Placement in Computing Continuum for a Robotic Application

Zeinab Bakshi[1*], Zahra Najafabadi Samani[2], Guillermo Rodriguez-Navas[1], Hans Hansson[1] and Radu Prodan[3]

[1]Department of Innovation, Design, and Engineering, Mälardalen University, Sweden.
[2]Distributed and parallel system group, University of Innsbruck, Austria.
[3]Department of Information Technology, University of Klagenfurt, Austria.

*Corresponding author(s). E-mail(s): Zeinab.Bakhshi@mdu.se;
Contributing authors: Zahra.Najafabadi-samani@uibk.ac.at;
Guillermo.Rodriguez-navas@mdu.se; Hans.Hansson@mdu.se; Radu.Prodan@aau.at;

Abstract

This paper analyzes the timing performance of a persistent storage designed for distributed container-based architectures in industrial control applications. The storage ensures data availability and consistency while accommodating faults. The analysis considers four aspects: 1. placement strategy, 2. design options, 3. data size, and 4. evaluation under faulty conditions. Experimental results considering the timing constraints in industrial applications indicate that the storage solution can meet critical deadlines, particularly under specific failure patterns. Moreover, this evaluation method is applicable for assessing other container-based critical applications with timing constraints that require persistent storage. Further comparison results reveal that, while the method may underperform current centralized solutions under fault-free conditions, it outperforms the centralized solutions in failure scenarios.

Keywords: Fault-tolerance, persistent storage, cloud, fog, edge, computing continuum.

1 Introduction

Providing reliable and persistent storage in container-based architectures is an ongoing research challenge [1–3]. Such services are critical for stateful applications that need reliable storage and immediate availability. Despite ongoing research efforts, the provision of persistent storage in container-based architectures remains an active area of investigation.

Various cloud-native methods and tools are available for deploying stateful applications in orchestrated container architectures. For example, [1] uses a deployment controller, while [4] involves third-party solutions such as Ceph storage. For container-based fog architectures, there is a need for local persistent storage that ensures data consistency and availability, particularly for time-sensitive stateful applications [5]. While existing solutions address these challenges, they fail to provide an effective storage system for stateful applications that meet timing constraints in a computing continuum architecture that spans all system layers from cloud servers to edge sensors and actuators [6].

To address this, we expand our prior study [5], which leverages a Replicated Data Structure (RDS) for local state storage on each device in the

cluster together with a Storage Container (SC) for data propagation and consistency. This solution offers data management, local state storage, failure resilience, and data consistency through the RAFT [7] consensus protocol. We present an experimental study of a robotic application [5] to evaluate how different storage placements in the computing continuum layers [8] meet the application runtime requirements. We aim to determine the most suitable storage deployment tailored to the specific application, addressing four research questions:

RQ1: How does the storage at different computing continuum layers impact the application response time?

RQ2: How do centralized or distributed storage designs affect the application response time?

RQ3: How does varying data size affect the response, access, and exchange times?

RQ4: How do different failure types impact response time?

Specifically, our contributions are as follows:

1. We evaluate the runtime performance of our solution [5] deployed at each layer of the computing continuum architecture [8].
2. We evaluate application response time with varying data access time and location under different centralized and distributed designs.
3. We provide insight into using our solution for different application timing and data size requirements.

Outline. This article has eight sections. We give a background on the considered robotic application and its tolerable response time in Section 2. Section 3 reviews related work and identifies the most relevant ones with respect to our comparison criteria. We describe the resources and application model in Section 4, followed by presentation of two storage design options in Section 5. We explain the four-dimensional design of our experiments in Section 6. Section 7, addressee the research questions and discusses the results. Finally,in Section 9, we conclude our work and outline possible future directions.

2 Robotic Application

We study a stateful robotic use-case application that avoids obstacles while moving toward its goals, implemented using the Robot Operating System (ROS) [9]. Like most control applications, the robotic application relies on exchanging messages, known as *topics* in ROS. We refer to these messages, as states in our studies, representing message history stored for future use by the same or other applications. We deployed this robotic application within a container-based architecture using Docker, as outlined in [5].

This robotic application has timing requirements, and we are interested in understanding its *tolerable response time*. The tolerable response time of a robot depends on various factors such as its speed, acceleration, sensing range, actuation and sensing intervals, and the vicinity of obstacles. A formula by Liu et al. [10] bounds the tolerable response time to the robot' sense-to-act time ($t \leq T$):

$$T = \frac{2 \cdot a \cdot d - v^2}{2 \cdot v \cdot a}, \tag{1}$$

where v is the robot's maximum velocity, a is its maximum acceleration, and d is the sense range.

We derive the tolerable response time using a specific robotic scenario with a maximum velocity of 4.98 m/s, maximum acceleration of 5 m/s^2, and sensing range of 4.5 m. The resulting sense-to-act time of 0.9 s represents the tolerable response time, $t \leq 0.9$ s, used as a *threshold* in our experiments for comparison to other solutions.

3 Related Work

This section reviews existing storage solutions for stateful applications in container-based architectures, summarized in Table 1.

3.1 Storage systems for container-based architectures

Volatile storage for stateful applications. Sharma et al. [20] proposed a distributed storage system using an application deployment on Kubernetes [23]. In contrast, Kristiani et al. [24] proposed a persistent volume for containerized applications deployed on the Kubernetes cluster on an OpenStack platform. However, these works do not consider data consistency in a distributed network. Additionally, the persistent storage system in these studies and other related ones [1, 4, 13] resides in the cloud layer with a high response time for time-sensitive and safety-critical applications control applications. To

Table 1: Summary of related works on container-based storage systems [11].

Feature	*Design*	*Data replication*	*Data consistency*	*Fault-tolerant storage*	*Fault tolerance*	*Self-healing*	*Timing analysis*	*Continuum layer*	*Application*	*Nodes*
[5]	**Distributed**	✓	✓	✓	✓	✓	✓	**Fog**	**Robotic**	≤ 70
[1, 12]	Centralized	—	—	—	✓	—	✓	Cloud	Video streaming	2
[2]	—	✓	✓	—	—	—	✓	Cloud	Log producer	4
[4]	Distributed	✓	—	—	—		✓	Cloud	*Unspecified*	3
[13]	—	✓	✓	—	—	—	✓	Cloud	Log producer	4
[14]	Distributed	✓	—	Flocker	✓	—	—	Edge	Hadoop	3
[15]	—	✓	✓	—	✓	—	✓	Cloud	Log producer	4
[16]	Centralized	—	—	—	✓	—	✓	Cloud	Video streaming	4
[17]	Centralized Distributed	✓	✓	—	✓	—	✓	Fog	Log producer	20
[18]	Centralized	—	—	—	✓	—	✓	Fog	VDCN	5
[19]	Distributed	✓	✓	✓	—	—	—	Cloud	E-commerce	10
[20]	Distributed	✓	—	Ceph	—	—	—	Cloud	—	—
[21]	Distributed	✓	✓	—	—	—	—	Cloud	—-	4
[22]	Distributed	—	—	—	—	—	✓	Cloud	—	—

provide data storage for edge applications, Ismail et al. [14] evaluate Docker Swarm for edge computing considering four criteria: deployment and termination, resource and service management, fault tolerance, and caching. However, they did not propose a unified fault tolerance mechanism and ignored application reintegration.

Fault tolerant storage. Designing a distributed, fault-tolerant storage system at the fog, edge, and cloud layers requires two main considerations [25]:

Persistent data storage that supports reintegration and recovery after failure while optimizing redundancy allocation in resource-constrained networks;

Distributed data consistency using consensus protocols appropriate for the system requirements and the complexity of the protocol.

For example, Netto et al. address the volatile storage issue by proposing the use of state-machine replication in containers [2], incorporating the RAFT consensus protocol in Kubernetes [15]. Although the level of data protection against container failure is high compared to other studies in the literature, the overhead increases the container footprints, changing their lightweight characteristics heavier than expected. This overhead is contrary to their lightweight nature, suitable at resource-constrained fog and edge layers, which reduces resource consumption and response time.

Contribution. The need for a persistent storage system to manage stateful applications and ensure timely data access at the fog layer arises from our robotic use-case [25]. We propose container-based storage applications that offer a distributed storage system and recover from failure by adopting RAFT protocol fulfilling correct data delivery, safety, liveliness, and fault-tolerance properties [11, 26]. Table 1 explores existing container-based storage solutions in the computing continuum, compared to our method.

Table 2: Failure and timing analysis papers.

	Centralized	*Distributed*
Cloud	[12] [1, 16]	[14] [15]
Fog	[17] [18]	[5] [17]

3.2 Criteria-based selection

To compare our proposed solution with related works, we categorize the storage solutions into four categories, illustrated in Table 2. We filter out less relevant solutions that do not meet our comparison criteria, such as timing performance and level of fault tolerance, and exclude papers that do not consider node failure recovery and timing analysis. In addition, a thorough comparison is impossible unless enough metrics and data are available. Therefore, we select the papers that fulfill these two requirements for the analysis. In the following, we explain why we choose the papers highlighted in green.

Centralized cloud storage. Several works [1–4, 12, 16, 19] provided persistent storage for stateful applications deployed using container-based architectures, as described in Section 3.1. Among them

works, Vayeghani et al. [1, 12, 16] consider both timing analysis and node recovery after failure. We choose the latest work [12] for further evaluation since, as an extension of the previous works [1, 16].

Distributed cloud storage. From two papers falling into this category, we choose the work by Netto et al. [15], as an extension of [3].

Centralized fog storage. The work of Ismail et al. [14] does not provide timing measurements, and thus, we do not consider it. Johansson et al. [18] implemented persistent volume placed in the master node of the Kubernetes cluster at the fog layer and simplified the assumption of a failure to the master node. Denzler et al. [17] compared different storage solutions at the fog layer, including the timing performance of centralized and distributed storage solutions on physical devices. Therefore, this work falls into centralized and distributed storage at the fog layer.

Distributed fog storage. We select our previous work [5] that includes the timing performance of distributed storage at the fog layer, extended in this paper to the other layers (cloud and edge). While our solution outperforms the centralized cloud solutions, the centralized storage at the fog layer needs further investigation.

4 Model

We need a suitable representation of the involved entities to understand the conceptual construct of the computing continuum architecture [8]. We provide a solution model in this section, comprising related components, resources, and interactions. For the sake of clarity, Table 3 summarizes the notation used in the paper.

4.1 Computing continuum

We model the computing continuum across three layers, Cloud, Fog, and Edge [27], differentiated by location, computing power, latency, and bandwidth, explained in the following paragraphs.

Cloud layer. The cloud is at the top of the computing continuum and has high-performance resources. However, they have high latency and low bandwidth, best suited for computing-intensive services that are not time-sensitive.

Table 3: Model notations.

	Notation	*Definition*
Application	$\mathcal{A}$	Set of applications
	A_i	Application
	m_{Ai}	Application data size
	A_i^P	Application processing demand A_i
	A_i^M	Application memory demand A_i
	A_i^S	Application storage demand A_i
Infrastructure	C_j	Computational resources
	$\mathcal{C}_D/\mathcal{F}_D/\mathcal{E}_D$	Physical and virtual machines in Cloud/Fog/Edge
	n_{jq}	Network connection between C_j and C_q
	L_{jq}/B_{jq}	Latency/bandwidth between C_j and C_q
	C_j^P	Processing capacity of C_j
	C_j^M	Memory capacity of C_j
	C_j^S	Storage capacity of C_j
Time	e_{A_i}	Execution time of an application
	$\tau(j,q)$	Data transmission time between C_j and C_q
	$\mathcal{E}$	Data exchange time (distributed)
	F_{SC_i}	*SC* fetch delay
	R^l	Raft protocol delay
	W_{SC_i}	*SC* write delay
	$\mathcal{E}^f$	Failure overhead on data exchange time
	N_{C_j}	Startup delay of C_j
	d_{SC_i}	Deployment latency of SC_i
	$\alpha/\lambda/\beta$	Maximum number of SC/ device/ application failures
	$r^{\mathcal{C}}_{A_i}/r^{\mathcal{D}}_{A_i}$	Application response time (centralized/distributed)
	R_{A_i}/W_{A_i}	Application read/write delay
	$r^{\mathcal{C}f}_{A_i}/r^{\mathcal{C}f}_{A_i}$	Failure overhead (centralized/distributed)
	$r^{wc-\mathcal{D}}_{A_i}$	Application worse-case response time (distributed)

Fog layer. The fog consists of storage and computing resources located between the cloud and the edge layers. These resources are in smaller facilities with limited capacity but offer higher bandwidth and lower latency than the cloud.

Edge layer. The edge includes small computing resources located between the fog and physical IoT devices like sensors and actuators. These computing resources are directly connected to the sensors and actuators, providing faster response times and greater data transfer speeds than the fog layer.

Two sets characterize these computing layers:

Computational resources. $C = \{C_1, \ldots, C_n\}$ refer to physical and virtual machines that host applications, commonly called devices. They have a triplet of resources $C_j = \left[C_j^P, C_j^M, C_j^S\right]$, where C_j^P represents the processing speed [1] (in a million instructions per second (MIPS)), C_j^M represents the computer's Random Access Memory (RAM) size (in MB), for managing processed data, and C_j^S denotes the storage capacity (in MB). Tools used for measuring MIPS, bandwidth and latency are specified in Section 6.

[1] We presume that the measurements have been normalized over time and under varying workloads, ensuring that processing speeds and other factors are comparable, even when there are variations in instruction sets and architecture.

Network connections. $\mathcal{N} = \{n_{jq} \| \ 1 \leq j, q \leq n\}$ between the computational resources is a set of tuples $n_{jq} = (\mathcal{L}_{jq}, B_{jq})$, where $\mathcal{L}_{jq}$ represents the delay and B_{jq} the bandwidth between the devices C_j and C_q.

4.2 Application model

We model a containerized robotic application developed in a ROS based on the use-case studies described in [5]. An application runs on different devices (i.e., fog, cloud) and interacts with other applications to provide a well-defined function, such as moving toward a conveyor belt.

Let $\mathcal{A} = \{A_1, A_2, A_3, \ldots, A_n\}$ be a set of robotic applications.

Robotic application. Every application $A_i \in \mathcal{A}$ has a triplet of computational demands $A_i = \left[A_i^P, A_i^M, A_i^S\right]$, where A_i^P represents the application's processing load (in millions of instructions (MI)), A_i^M represents the required memory (in MB), and A_i^S represents the required storage (in MB). The storage demand depends on the application size, the total data it generates, and the number of application iterations.

Runtime model. Applications have two runtime metrics, deployment and execution times, that vary depending on the hosting device. This simplified notation helps analyze the characteristics of individual applications without excessively representing specific device associations.

Deployment time. d_{A_i} of an application A_i represents the duration of the startup of scripts and initialization of the application.

Execution time. e_{A_i} of an application A_i on a device C_j is the ratio between its processing load and the device's processing speed.

$$e_{A_i} = \frac{A_i^P}{C_j^P}. \tag{2}$$

Execution model. Each application A_i runs for a defined number of iterations. After each iteration of execution, applications write their state m_i of a size given in the application specification to the defined storage location, as explained in Section 5.

Data transmission time. τ_{jq} between two devices C_j and C_q aggregates the link delay $\mathcal{L}_{jq}$ and the ratio of the message size m_i to the network bandwidth B_{jq} between the device C_j hosting an application and the device C_q storing the state:

$$\tau_{jq} = \mathcal{L}_{jq} + \frac{size\,(m_i)}{B_{jq}}. \tag{3}$$

5 Storage Design and Modelling

In this section, we present two storage design models evaluated in the context of our robotic application:

Distributed storage based on our previous solution [5] and extended by placing the storage at different layers of the computing continuum;

Centralized storage inspired by existing cloud solutions [17, 28], implemented and compared when located at different computing continuum layers.

5.1 Distributed storage

We implement a persistent storage space called *Replicated Data Structure (RDS) and Storage Containers (SC)* that manages data of RDS and is combined with the RAFT consensus protocol. The storage is distributed at all devices in the cluster while the applications are deployed at the edge and fog layers.

Design. Figure 1 illustrates the structure of the design. Applications write the state variables in a local RDS storage available on the device. The RDS stores the state variables of the application in the cluster in each device and keeps the replicas consistent. A state update must always propagate to all the devices in the cluster, with the help of SCs, so that other applications can access it. SCs automatically restart after a failure (as applications) and take responsibility for propagating state updates in the cluster. Still, applications do not block from reading and executing while data propagates through the devices. One SC, elected as a leader, is responsible for data consistency in a cluster of devices. For any state update on devices, the corresponding SC sends the updates to the leader, which broadcasts the updates to the other SCs. Thus, any changes in a state variable propagate to the whole cluster in a bounded time.

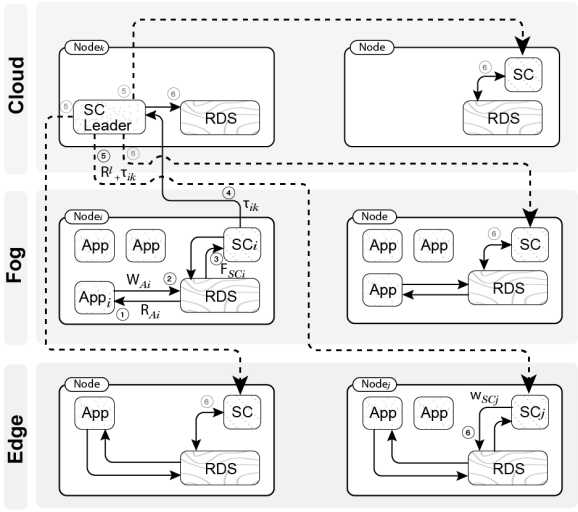


Fig. 1: Distributed computing continuum storage.

Data exchange time. $\mathcal{E}$ is the time required by one state value to be propagated to the system as:

$$\mathcal{E} = F_{SC_i} + \tau_{jq} + R^l + \max_{C_{j,q} \in C} \tau_{jq} + \max_{SC_i \in SC} \{W_{SC_i}\}, \tag{4}$$

F_{SC_i} is the time required by an SC to fetch data from a local RDS;
τ_{jq} is the time required to propagate the data transmitted to the SC leader to the cluster;
R^l is the Raft protocol delay. When all the SCs receive the state value from the leader,
W_{SC_i} is the time SC_i requires to write the state value in its local RDSs.
i,j,q are integer numbers, $1 \leq i, j, q \leq n$.

Failure overhead. $\mathcal{E}^f$ generated by a device or an SC failure during a data exchange adds a corresponding delay to the data exchange time:

$$\mathcal{E}^f = \max\left\{\lambda \cdot d_{SC_i} + \beta \cdot N_{C_j}\right\}, \tag{5}$$

d_{SC_i} represents the SC deployment time;
N_{C_j} represents the device startup time;
λ, β are the maximum number of SC and device failures that can occur on a device during the data exchange delay.

Application response time. $r^{\mathcal{D}}_{A_i}$ includes application writing, reading, deployment, execution, failure, and recovery time for re-execution. In fault-free conditions, the application response time is:

$$r^{\mathcal{D}}_{A_i} = R_{A_i} + d_{A_i} + e_{A_i} + W_{A_i} + \mathcal{E}, \tag{6}$$

R_{A_i}, W_{A_i} are the application read and write time from, respectively to the local storage;
d_{A_i}, e_{A_i} are the application deployment and execution times.

Failure overhead. to the response time is:

$$r^{\mathcal{D}f}_{A_i} = (\alpha + \beta + 1) \cdot r_{A_i} + \beta \cdot \left(N_{C_j}\right) + \mathcal{E}^f, \tag{7}$$

α, β are the number of application and device failures occurring at the worst possible time, immediately before completing the task;
r_{A_i} is application response time;
N_{C_j} is device startup time;
$\mathcal{E}^f$ is the failure overhead time.

Worst-case response time (WCRT). $r^{wc-\mathcal{D}}_{A_i}$ on failures is:

$$r^{wc-\mathcal{D}}_{A_i} = r^{\mathcal{D}}_{A_i} + r^{\mathcal{D}f}_{A_i}, \tag{8}$$

$r^{\mathcal{D}}_{A_i}$ is the application response time in distributed fault-free conditions;
$r^{\mathcal{D}f}_{A_i}$ is the corresponding failure overhead.

5.2 Centralized persistent storage

We implement a centralized solution at all three layers, cloud, fog, and edge, to help us: 1. compare the distributed solution in an extended fog and edge layers environment; 2. identify the possibilities and limitations of improving our solutions based on the findings of the experiments.

Design. Figure 2 illustrates the data flow using centralized storage, where applications directly write their state to the database located either at the same layer (see Figure 2-A) or at the cloud layer (Figure 2-B). We consider a replica for the database to avoid a single database point of failure. However, it is time-consuming and inefficient to redirect applications to the replica of the database to access their states. We use the timing parameters from [12] and [17] for storage re-integration in the case of cloud and fog locations, respectively.

Data access time. Θ is the time required by one state value to be accessible by the applications:

$$\Theta = R_{A_i} + W_{A_i} + 2 \cdot \tau_{jq}; \tag{9}$$

R_{A_i}, W_{A_i} represents the application read and write times from, respectively, to local storage.

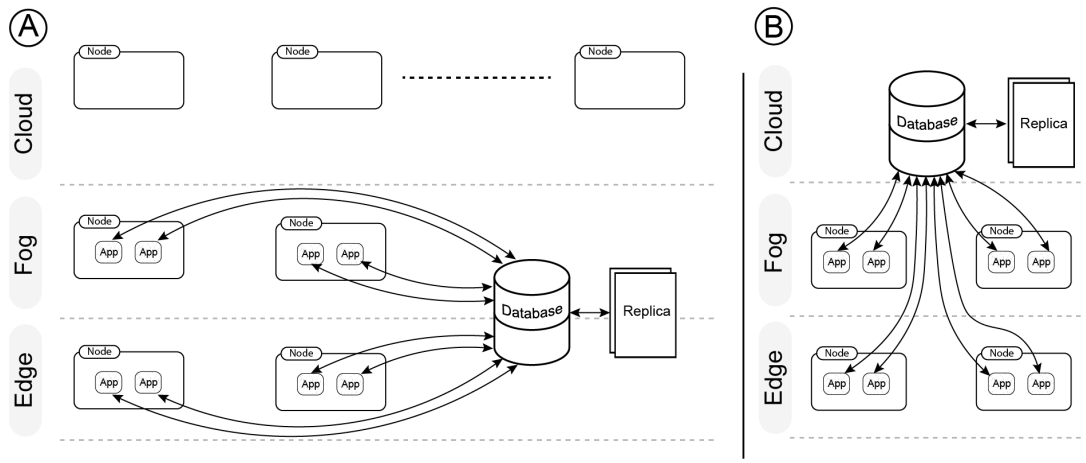


Fig. 2: Centralized persistent fault-tolerant storage in computing continuum.

τ_{jq} is the data transfer time between the application deployment device C_j and the centralized storage node.

Failure overhead. Θ^f accounts for the restart and data re-transmission time if an application or a node hosting it fails during data access:

$$\Theta^f = \max\left\{\lambda \cdot (d_{SC_i}) + \beta \cdot (N_{C_j})\right\}, \tag{10}$$

λ, β are the number of SC and device failures;
d_{SC_i} is the SC deployment time;
N_{C_j} is device startup time;

Application response time. $r^{\mathcal{C}}_{A_i}$ in fault-free settings is:

$$r^{\mathcal{C}}_{A_i} = d_{A_i} + e_{A_i} + \Theta, \tag{11}$$

d_{A_i}, e_{A_i} are the application deployment and execution times.
Θ is the read and write time to the centralized storage in the network, including the data transition time from the application running device to the storage.

Failure overhead. $r^{\mathcal{C}f}_{A_i}$ to the response time is:

$$r^{\mathcal{C}f}_{A_i} = (\alpha + \beta + 1) \cdot r^{\mathcal{C}}_{A_i} + \beta \cdot N_{C_j} + \Theta^f, \tag{12}$$

α, β are the number of application and device failures occurring at the worst possible time, immediately before completing the task;
N_{C_j} is device startup time;
Θ^f is the failure overhead on data access time.

Worst-case response time (WCRT). of an application response accounts for the failure time of the entities involved, calculated when using centralized storage as follows:

$$r^{wc-\mathcal{C}}_{A_i} = r^{\mathcal{C}}_{A_i} + r^{\mathcal{C}f}_{A_i}. \tag{13}$$

To meet the application requirements, the worst-case response time must be shorter than the tolerable response time.

6 Experimental Design

This section outlines our experimental design using an in-house discrete event simulator developed in Python [2]. We examine the impact of storage placement, storage design, data size, and failures on application response time, as introduced in the research questions in Section 1.

6.1 Infrastructure design

We run experiments on the computing continuum infrastructure distributed across the cloud, fog, and edge layers, as summarized in Table 4.

Cloud layer. The cloud contains virtualized instances provisioned on-demand on high-end data centers in AWS in Virginia, Google in Frankfurt, and Exoscale in Sofia.

[2]Simulation files are available on a GitHub repository: https://github.com/ZeinabBa/Comparison-Persistent-Storage

Table 4: Computing continuum testbed.

Computing layer	*Cloud*			*Fog*			*Edge*		
Provider	AWS	Google	A1 / Exoscale		Klagenfurt		University of Klagenfurt		
Instance type	`t2.xlarge`	`n2.standard-4`	`large`	`medium`	`medium`	`large`	*NJN*	*RPi4*	*RPi3*
Location	Virginia	Frankfurt	Sofia	Klagenfurt, Vienna, Munich	Klagenfurt		Klagenfurt		
Processing speed [MIPS]	11 200	10 100	14 000	7200	58 000	21 700	4080	5100	3500
Processing cores	4	4	4	2	12	8	4	4	4
Memory [GB]	16	16	8	4	32	16	4	4	1
Storage [GB]	8	8	10	10	32	32	16	16	16
Bandwidth [Mbit/s]	100–710	500–800	450–850				300–920		
Latency [ms]	15–100	10–30	7–28				1–2		

Table 5: Application specification.

Identifier	1	2	3	4	5	6	7	8	9	10
CPU speed [GHz]	1	2	1.1	3.2	1	1.4	1.2	2	1.3	1.6
Memory [GB]	2	3	3	4	1	4	2	4	3	4
Storage [GB]	2	4	3	3	0.8	2	3	3	1	2
Workload [MI]	1487	1823	1851	2542	2073	2591	1758	2261	1971	2480
Iterations [No]	2	3	4	2	4	4	5	3	4	4

Fog layer. The fog contains virtualized instances provisioned on-demand from the Exoscale provider, and bare-metal instances at the University of Klagenfurt [8].

Edge layer. The edge consists of NVIDIA Jetson Nano (*NJN*) running `Linux for Tagra`, Raspberry Pi-3 (*RPi3*), and thirty Raspberry Pi-4 (*RPi4*) with `Raspberry Pi OS` at the University of Klagenfurt.

6.2 Application design

We define the robotic applications based on the ROS [28] specifications, comprising their resource demands, tolerable response time, and number of iterations for each application as summarized in Table 5. We deploy 250 applications for each experiment based on the scalability analysis performed in previous work [29]. We set the ratio of nodes hosting the applications based on their resource demands. We measure MIPS using the `p7zip` tool and the latency and bandwidth using the `iperf` and `icmp echo` tools at average network load [3].

6.3 Storage placement (RQ1)

We defined two scenarios to investigate the impact of storage placement at different computing continuum layers on the response time regarding RQ1. Moreover, they provide a better understanding to the other research questions.

[3]When the network is operating without any unusual spikes or extreme levels of traffic or usage.

Table 6: Percentage of cloud, fog and edge resources in dispersed deployment scenarios.

Scenario	*Cloud* [%]	*Edge* [%]	*Fog* [%]
CLOUD-2	2	49	49
CLOUD-7	7	46	47
CLOUD-11	11	44	44

Colocated deployment. This scenario examines the timing performance when placing the applications and storage in the same layer, as follows:
EDGE application and storage at the edge.
FOG application and storage in the fog.

Dispersed deployment. This scenario examines the impact of cloud storage on the application performance, evenly scattered across both the edge and fog layers, using the percentage of devices presented in Table 6. We limit the use of cloud devices to 11 %, which is sufficient for storing the data sizes used in our experiments.

6.4 Storage design (RQ2)

To address RQ2, we used the same scenarios as in Section 6.3 with two storage design variations: *centralized* and *distributed* (see Section 5).

6.5 Storage data size (RQ3)

To address RQ3, we use the scenarios in Section 6.3 to analyze the effect of data size on the response time, incrementally increased in five steps: 64 kB, 512 kB, 2 MB, 5 MB, and 10 MB.

6.6 Failure Scenarios (RQ4)

To examine the impact of failures on response time (RQ4), we design three types of failures.

Fault-free serves as a baseline comparison.

Random failures injected into the system at arbitrary times follow a normal distribution with an upper limit of 10 % of cluster components.

Specific failure corresponds to worst-case scenarios for distributed and centralized storage.

Distributed storage. We identify specific faults through a formal verification explained in [30]. For instance, recovery after failure takes longer when a leader fails, requiring a leader election.

Centralized storage. We simulate specific failures by introducing faults where a single storage system is responsible for data management. Despite the presence of a replica for the centralized storage, we consider it a potential single point of failure, as the time required to take over the primary storage result in an increased overhead.

7 Evaluation

As the basis for answering the research questions introduced in Section 1, we evaluate three key metrics, response time, data exchange time, and data access time, under the conditions and scenarios outlined in Section 6. We format the results using a uniform graph representation as follows:

X-axis represents the data sizes.
Y-axis represents the time in milliseconds.
Bars represent the response times.
Red dashed lines indicate the threshold for the tolerable response time.
Line charts display data access and exchange times.
Side bars differentiated by colors present response times in centralized and distributed designs.
Overlaying bars show the response times in baseline scenarios and under failure conditions.

We conclude each experimental analysis with several insights.

7.1 RQ1: Impact of storage placement on response time

We address RQ1 by examining the collocated and dispersed deployment scenarios outlined in Section 6.

Colocated deployment. The results for colocated deployment are shown in Figure 3. We observe that the application response time for deploying applications and accessing storage at the edge layer exceeds the tolerable threshold by 35.6 %. However, allocating the applications and the storage in the fog layer significantly reduces the response time by 80.4 % compared to the edge, which is 63.4 % below the threshold. We further observe that data exchange and access times are relatively low. The high response time experienced at the edge layer is primarily due to the relatively high application deployment and execution time caused by limited resources. These findings highlight the importance of considering storage placement in the fog layer and leveraging its resources to achieve tolerable response times.

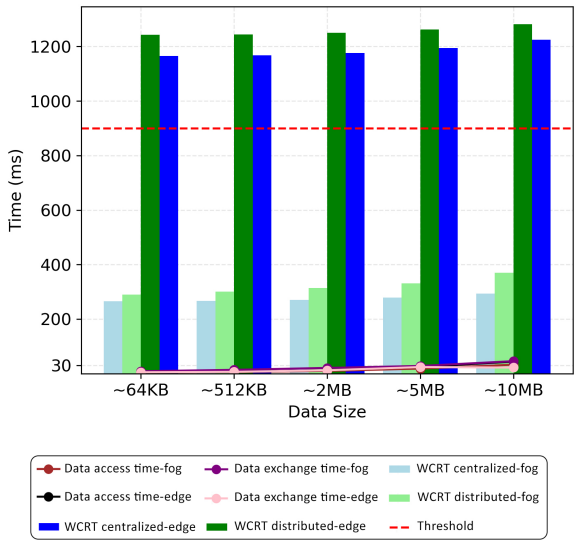
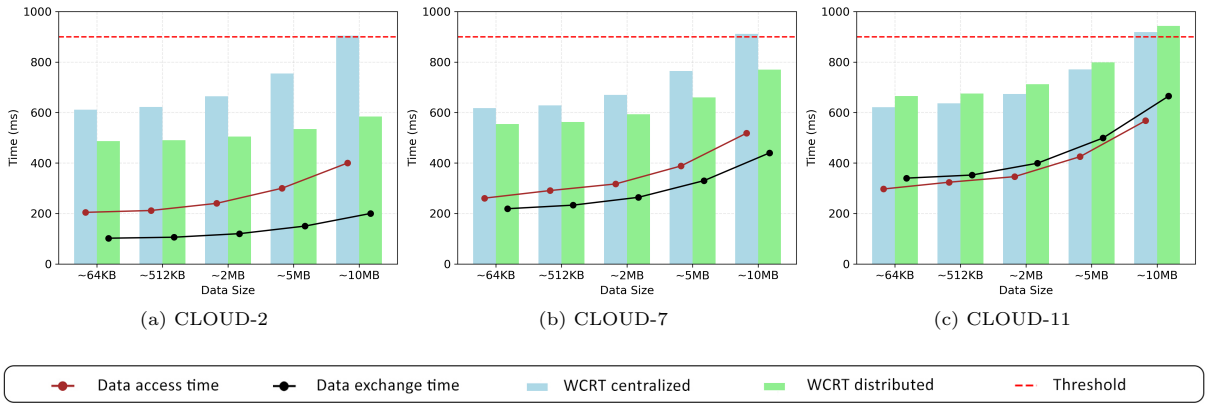


Fig. 3: Collocated deployment performance results.

Dispersed deployment. The dispersed deployment results in Figure 4 highlight that utilizing more cloud nodes increases response time due to increased communication time, as indicated by data exchange and data access times. For example, in the CLOUD-2 scenario shown in Figure 4a, the response time for a data size of 64 kB is 486.7 ms for centralized and 611.1 ms for distributed storage. Notably, this impact is slightly higher on the distributed design than on the centralized design. In the subsequent subsections, we will elaborate on various factors affecting timing performance in a dispersed deployment scenario.

Concluding insight. Using the edge layer to place applications and storage results in an intolerable response time for all data sizes, even in fault-free conditions and regardless of its centralized or distributed design. The high application execution time imposes significant overhead on the overall



(a) CLOUD-2 (b) CLOUD-7 (c) CLOUD-11

Fig. 4: Dispersed deployment performance results.

response time, primarily due to the limited computational resources at the edge layer. Therefore, we do not recommend deploying applications and placing the storage solely at the edge.

7.2 RQ2: Impact of storage design on response time

To answer RQ2, we simulate colocated and dispersed deployment scenarios using distributed and centralized designs.

Colocated deployment. Figure 3 shows that the response time is slightly higher in the distributed design compared to the centralized design. In particular, we observe an average increase of 7 % in response time for applications colocated at the edge and a 16.4 % increase in the fog using the distributed design compared to the centralized one. These results correspond to failure-free conditions and may vary in the presence of failures. The lower performance of the distributed storage at both edge and fog layers is due to the overhead caused by data exchange within the consensus algorithm. However, distributed and centralized designs have relatively low data exchange and access times.

Dispersed deployment. Figure 4 shows that the distributed storage decreases the response time by an average of 36.7 % by placing the distributed storage leader in the cloud, in contrast to the centralized design that places both the storage and its replica in the cloud. However, this trend changes as the number of cloud storage nodes increases. Figures 4b and 4c reveal that increasing the number of cloud resources to 7 % and 11 % increases the response time by 21 % and 46.2 % compared to CLOUD-2 scenario. Meanwhile, the centralized design demonstrates relatively stable performance without significant changes. In the CLOUD-11 scenario, the centralized design provides a 4.3 % lower response time than the distributed design.

Concluding insight. Leveraging a centralized storage design demonstrates superior timing performance at both edge and fog layers compared to the distributed design.

Based on the obtained results, we conclude that increasing the ratio of cloud resources for data storage in the distributed design significantly increases the data exchange time. Consequently, it worsens the response time due to the consensus algorithm, necessitating distributing states over the entire cluster for each data.

7.3 RQ3: Effect of data size on response time

We report colocated and dispersed scenario results while varying the application data size for centralized and distributed storage designs.

Colocated deployment. Intuitively, as the data size increases, the response time increases in all scenarios for centralized and distributed storage designs. The response time at the edge becomes intolerable even for the smallest data size of 64 kB. Scaling the data to 10 MB increases the response time at an average rate of 5 % and 3.1 % in

response time for the centralized and distributed designs. In the fog, the response time increases by 22.2 % and 10.5 % from the initial 64 kB data size to 10 MB in the centralized and distributed designs. However, this increase is not drastic.

Dispersed deployment. As the data size increases, the response time increases across all dispersed deployment scenarios. However, the response time for different data sizes in the distributed CLOUD-2 and CLOUD-7 designs remains tolerable. On the other hand, the response time exceeds the threshold only at 10 MB data size in the centralized design. The situation changes in the CLOUD-11 scenario when the response time becomes intolerable at the 10 MB data size for both distributed and centralized designs due to the latency between different computing continuum layers.

Concluding insight. As data size grows, the response time increases. It is worth noting that the increase in response time is not drastic, as observed within the range of data sizes from 64 kB to 10 MB. Nevertheless, utilizing cloud resources for data storage results in higher response times as data size grows compared to placing the storage at fog layer. This can be attributed to the latency between different layers of the computing continuum architecture.

7.4 RQ4: Influence of failures on response time

We analyze the performance of the robotic application in the presence of failures using the worst-case response time under fault-free conditions as a baseline. We exclude the colocated EDGE scenario from our analysis since its worst-case response time exceeds the tolerable threshold even in fault-free conditions.

Random failures.

FOG. Figure 5a illustrates that the WCRT of the application colocated in the fog remains considerably lower than the acceptable threshold despite an average increase of 69.8 % and 16 % for centralized and distributed designs. The centralized storage design showcases superior performance despite random failure. However, the results indicate that the response time remains significantly below the threshold, and the impact of failure on both the centralized and distributed designs is relatively comparable.

CLOUD-2. Figure 5b shows that the application's worst-case response time remains acceptable for both centralized and distributed designs for a data size below 2 MB. However, the response time exceeds the threshold for data sizes larger than 5 MB in the centralized design and for data sizes larger than 10 MB in the distributed design.

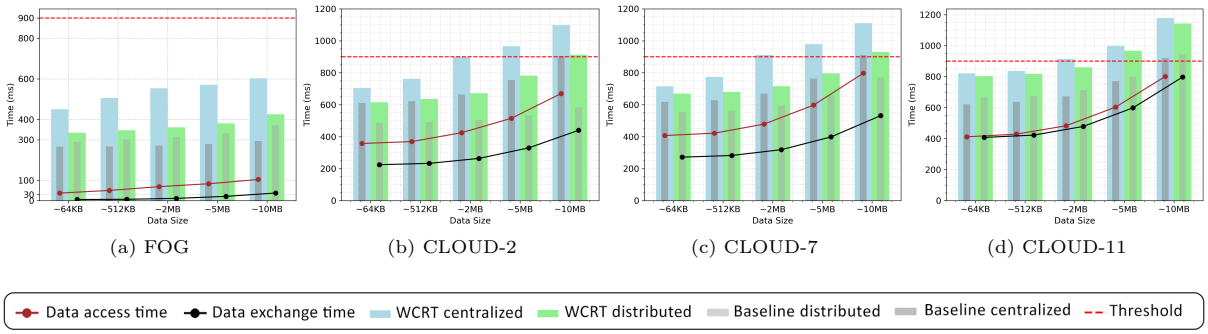
CLOUD-7. The response time in the centralized design exceeds the threshold for data sizes of 2 MB and larger. Similar to *CLOUD-2*, the response time in the distributed design surpasses the threshold for data sizes larger than 10 MB. The average increase of 20 % and 11 % in the worst-case response time for centralized and distributed designs, respectively, compared to the *fault-free* experiments. The distributed storage design demonstrates improved performance.

CLOUD-11. The situation differs for this scenario. As expected, based on the fault-free condition results of this scenario, the response time for the distributed design is slightly higher upon random failures. Consequently, There is an average increase of 32 % and 20 % in the worst-case response time for centralized and distributed designs, which exceeds the acceptable response time for data sizes larger than 2 MB.

Specific failures.

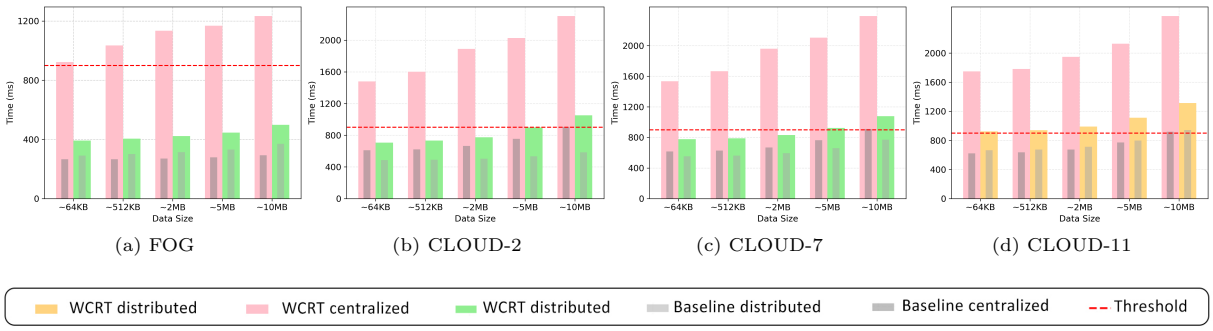
Colocated FOG. As depicted in Figure 6a, the worst-case response time of the distributed design remains tolerable for all data sizes. However, despite exhibiting better performance in fault-free conditions, the centralized design experiences a significant decline upon specific failures, exceeding the acceptable response time for all data sizes.

Dispersed CLOUD. Despite adding a replica in the centralized design to enhance data availability, specific failures in the centralized database or its replica significantly impact the timing performance. Figures 6b, 6c, and 6d illustrate that the worst-case response time of the centralized design exceeds the tolerable response time in the presence of specific failure for all scenarios and data sizes. In contrast, the response time increase remains within the acceptable range for data sizes below 2 MB for the distributed design.



(a) FOG (b) CLOUD-2 (c) CLOUD-7 (d) CLOUD-11

Fig. 5: Response time in the presence of random failures.



(a) FOG (b) CLOUD-2 (c) CLOUD-7 (d) CLOUD-11

Fig. 6: Response time in the presence of specific failures.

Concluding insight.

Fault-free execution. In fault-free conditions, centralized storage generally outperforms the distributed design in most scenarios. The reason is the synchronization overhead in the distributed design, particularly when locating the leader in the cloud layer, and the number of nodes increases, leading to an increased data transfer time.

Faulty execution. The results change when running applications with random and specific failures. For all scenarios, the distributed design performs better than the centralized design. Nevertheless, the centralized design maintains performance within the tolerable threshold in both a colocated fog deployment and a dispersed deployment for data sizes up to 2 MB. On the other hand, specific failures introduce a drastic increase in response time for the centralized design, rendering it intolerable for all scenarios. In the distributed design, however, even in the presence of specific failure, the response time remains well below the threshold when placing applications and storage at the fog layer for data sizes up to 2 MB.

8 Justification of the Approach

We built our simulation upon the real testbed infrastructure presented in Table 4 and replicated the device resources, communication patterns, and workloads using parameters obtained from the actual setup. Additionally, we incorporate application specifications and allocation algorithms derived from measurements. The choice of using a simulator instead of the actual setup derives from three primary reasons.

Iterative and optional design. We assess the feasibility of our fault-tolerant distributed storage system through a comparative analysis, encompassing four dimensions as explained in Section 1. Each dimension introduces a complexity layer within the iterative design process and the range of available design choices. Using simulation-based

evaluation helps us delve deeper into the design and implementation of various comparison dimensions. For example, simulation facilitates comparing our approach and a centralized design. Implementing the latter in an actual environment could be expensive and time-consuming.

Data generation for comprehensive evaluation. To thoroughly evaluate our system, we need to gather more data involving exploring diverse scenarios and generating substantial data as the foundation for analysis. Simulations help us by testing different scenarios and generating sufficient data for analysis, which is challenging to collect from real experiments.

Long-term studies and collaboration. Conducting extended studies and collaborating effectively can be complex when dealing with the actual infrastructure testbed demanding physical presence and continuous maintenance, potentially causing interruptions in its original purpose. Simulation allows us to conduct experiments conveniently without geographical limitations. Simulating the setup also provides a platform for other researchers to access and utilize it as a foundation for their investigations. This approach promotes a collaborative research environment and facilitates easy accessibility for interested researchers.

Validation. Leveraging simulation based on the actual setup also enables us to validate the accuracy of our results, achieved by comparing the simulation outcomes with sample real testbed tests. This comparison process ensures that our simulation faithfully represents the real system's behavior, enhancing the trust in its accuracy.

9 Conclusion

We have evaluated a close-to-edge persistent fault-tolerant storage for a robotic architecture in container-based architectures. We assess its behavior with increasing data size and compare it to alternative centralized design options. We also explore the impact of different placement strategies within the computing continuum framework under random and specific failure scenarios. Our primary objective is to compare the timing performance of our solution against existing alternatives. The evaluation results demonstrate that the distributed persistent storage design outperforms the current centralized design alternatives when facing random and specific failures.

In future research, we aim to expand our investigations to a wider range of applications. Moreover, we will carry out real-world experiments to further validate the simulation results and the approaches proposed in this work.

Ackowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement 764785 (FORA); VINNOVA project 2018-02437; and rant agreement 101016835 (DataCloud).

References

[1] L. Abdollahi Vayghan, M. A. Saied, M. Toeroe, and F. Khendek. Microservice based architecture: Towards high-availability for stateful applications with kubernetes. In *IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, pages 176–185, 2019.

[2] Hylson V Netto, Lau Cheuk Lung, Miguel Correia, Aldelir Fernando Luiz, and Luciana Moreira Sá de Souza. State machine replication in containers managed by kubernetes. In *Journal of Systems Architecture*, volume 73, 2017.

[3] Hylson Vescovi Netto, Aldelir Fernando Luiz, Miguel Correia, Luciana de Oliveira Rech, and Caio Pereira Oliveira. Koordinator: A service approach for replicating docker containers in kubernetes. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 58–63.

[4] Lubos Mercl and Jakub Pavlik. Public cloud kubernetes storage performance analysis. In *International Conference on Computational Collective Intelligence*, pages 649–660. Springer, 2019.

[5] Zeinab Bakhshi, Guillermo Rodriguez-Navas, and Hans Hansson. Fault-tolerant permanent

storage for container-based fog architectures. In *Proceedings of the 2021 22nd IEEE International Conference on Industrial Technology (ICIT)*, 2021.

[6] Daniel Balouek-Thomert, Eduard Gibert Renart, Ali Reza Zamani, Anthony Simonet, and Manish Parashar. Towards a computing continuum: Enabling edge-to-cloud integration for data-driven workflows. *The International Journal of High Performance Computing Applications*, 33(6):1159–1174, 2019.

[7] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In {*USENIX*} *Annual Technical Conference ({USENIX}{ATC} 14)*, pages 305–319, 2014.

[8] Dragi Kimovski, Roland Mathá, Josef Hammer, Narges Mehran, Hermann Hellwagner, and Radu Prodan. Cloud, fog, or edge: Where to compute? *IEEE Internet Computing*, 25(4):30–36, 2021.

[9] Jason M O'Kane. A gentle introduction to ros. In *University of South Carolina*. Independently published, available at http://www.cse.sc.edu/ jokane/agitr/, 2014.

[10] Sikang Liu, Michael Watterson, Sarah Tang, and Vijay Kumar. High speed navigation for quadrotors with limited onboard sensing. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1484–1491, 2016.

[11] Zeinab Bakhshi. Persistent Fault-tolerant Storage at the Fog layer. *Licentiate Thesis, Mälardalen University*, 2021.

[12] Leila Abdollahi Vayghan, Mohamed Aymen Saied, Maria Toeroe, and Ferhat Khendek. A kubernetes controller for managing the availability of elastic microservice based stateful applications. *Journal of Systems and Software*, 175:110924, 2021.

[13] Caio Oliveira, Lau Cheuk Lung, Hylson Netto, and Luciana Rech. Evaluating raft in docker on kubernetes. In *International Conference on Systems Science*, pages 123–130. Springer, 2016.

[14] B. I. Ismail, E. Mostajeran Goortani, M. B. Ab Karim, W. Ming Tat, S. Setapa, J. Y. Luke, and O. Hong Hoe. Evaluation of docker as edge computing platform. In *2015 IEEE Conference on Open Systems (ICOS)*, pages 130–135, 2015.

[15] Hylson Netto, Caio Pereira Oliveira, Luciana de Oliveira Rech, and Eduardo Alchieri. Incorporating the raft consensus protocol in containers managed by kubernetes: an evaluation. In *International Journal of Parallel, Emergent and Distributed Systems*, volume 35, pages 433–453. Taylor & Francis, 2020.

[16] Leila Abdollahi Vayghan, Mohamed Aymen Saied, Maria Toeroe, and Ferhat Khendek. Kubernetes as an availability manager for microservice applications. *arXiv preprint arXiv:1901.04946*, 2019.

[17] Patrick Denzler, Daniel Ramsauer, Thomas Preindl, Wolfgang Kastner, and Alexander Gschnitzer. Comparing different persistent storage approaches for containerized stateful applications. In *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2022.

[18] Bjarne Johansson, Mats Rågberger, Thomas Nolte, and Alessandro V Papadopoulos. Kubernetes orchestration of high availability distributed control systems. In *2022 IEEE International Conference on Industrial Technology (ICIT)*, pages 1–8. IEEE, 2022.

[19] Haifeng Liu, Wei Ding, Yuan Chen, Weilong Guo, Shuoran Liu, Tianpeng Li, Mofei Zhang, Jianxing Zhao, Hongyin Zhu, and Zhengyi Zhu. Cfs: A distributed file system for large scale container platforms. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1729–1742, 2019.

[20] Ashish Sharma, Sarita Yadav, Neha Gupta, Shafali Dhall, and Shikha Rastogi. Proposed

model for distributed storage automation system using kubernetes operators. In *Advances in Data Sciences, Security and Applications*, pages 341–351. Springer, 2020.

[21] Nguyen Nguyen and Taehong Kim. Toward highly scalable load balancing in kubernetes clusters. *IEEE Communications Magazine*, 58(7):78–83, 2020.

[22] Amit Warke, Mohamed Mohamed, Robert Engel, Heiko Ludwig, Wayne Sawdon, and Ling Liu. Storage service orchestration with container elasticity. In *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, pages 283–292. IEEE, 2018.

[23] Kubernetes Foundation, Kubernetes Documentation. https://kubernetes.io/.

[24] Endah Kristiani, Chao-Tung Yang, Yuan Ting Wang, and Chin-Yin Huang. Implementation of an edge computing architecture using openstack and kubernetes. In *International Conference on Information Science and Applications*, pages 675–685. Springer, 2018.

[25] Zeinab Bakhshi, Guillermo Rodriguez-Navas, and Hans Hansson. Fault-tolerant permanent storage for container-based fog architectures. In *22nd IEEE International Conference on Industrial Technology (ICIT)*, volume 1, pages 722–729, 2021.

[26] Ali Shahaab, Ben Lidgey, Chaminda Hewage, and Imtiaz Khan. Applicability and appropriateness of distributed ledgers consensus protocols in public and private sectors: A systematic review. *IEEE Access*, 7:43622–43636, 2019.

[27] Vincenzo De Maio and Dragi Kimovski. Multi-objective scheduling of extreme data scientific workflows in fog. *Future Generation Computer Systems*, 106:171–184, 2020.

[28] Yukihiro Saito, Futoshi Sato, Takuya Azumi, Shinpei Kato, and Nobuhiko Nishio. Rosch:real-time scheduling framework for ros. In *IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 52–58, 2018.

[29] Zeinab Bakhshi, Guillermo Rodriguez-Navas, and Hans Hansson. Analyzing the performance of persistent storage for fault-tolerant stateful fog applications. *Journal of Systems Architecture*, page 103004, 2023.

[30] Zeinab Bakhshi, Guillermo Rodriguez-Navas, and Hans Hansson. Using UPPAAL to verify recovery in a fault-tolerant mechanism providing persistent state at the edge. In *26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–6, 2021.