
Run-time Application Migration using Checkpoint/Restore In Userspace

Aleksandar Tošić

*University of Primorska Faculty of Mathematics, Natural Sciences and Information Technologies, 6000 Koper, Slovenia
InnoRenew CoE, Livade 6a, 6310 Izola
E-mail: aleksandar.tosic@upr.si*

Received 21 August 2023; Accepted 14 April 2024

Abstract

This paper presents an empirical study on the feasibility of using Checkpoint/Restore In Userspace (CRIU) for run-time application migration between hosts, with a particular focus on edge computing and cloud infrastructures. The paper provides experimental support for CRIU in Docker and offers insights into the impact of application memory usage on checkpoint size, time, and resources. Through a series of tests, we establish that the time to checkpoint is linearly proportional to the size of the memory allocation of the container, while the restore is significantly lower. Our findings contribute to the understanding of CRIU's performance and its potential use in edge computing scenarios. To obtain accurate and meaningful findings, we monitored system telemetry while using CRIU to observe its impact on the host machine's CPU and RAM utilization. Although our results may not be groundbreaking, they offer a good overview and a technical report on the feasibility of using CRIU on edge devices, which are typically resource constrained. This study's findings and experimental support for CRIU in Docker could serve as a useful reference for future research on performance optimization and application migration using CRIU.

Keywords: Checkpoint/Restore, edge computing, run-time container migration.

Journal of Web Engineering, Vol. 23_5, 735–748.

doi: 10.13052/jwe1540-9589.2357

© 2024 River Publishers

1 Introduction

In recent years, there has been a significant shift in the way computing is done. Instead of relying solely on centralized cloud-based systems, there has been a move towards decentralized edge computing systems. Edge computing systems bring computing resources closer to the end-users, reducing latency, improving performance, verifiability [1] and enabling new use cases that were previously unfeasible. However, with adopting the edge computing paradigm, a new set of challenges are introduced, such as resource constraints, high degree of hardware variability, and unreliable connectivity.

Checkpoint/Restore In Userspace (CRIU) is a powerful tool that allows the migration of running applications from one host to another while preserving their states. This tool has significant potential for use in edge computing systems, where migration of running applications is necessary due to resource constraints and the need to move services closer to the inherently dynamic demand. The ability to migrate running applications while preserving their state can significantly improve the reliability, availability, and fault-tolerance of edge computing systems. A few potential use case pertaining to the aforementioned use case are:

1. **Fault tolerance and disaster recovery:** CRIU can be used to provide fault tolerance and disaster recovery capabilities in edge computing systems. By migrating running applications to a different host in the event of a failure or disaster, CRIU can help ensure that critical services remain available and reduce the impact of downtime.
2. **Mobility:** CRIU can be used to provide mobility capabilities in edge computing systems. By allowing running applications to be migrated between different hosts, CRIU can enable new use cases such as mobile edge computing, where services follow users as they move around.
3. **Dynamic scaling:** CRIU can be used to enable dynamic scaling of services in edge computing systems. By migrating running applications between hosts based on changes in demand, CRIU can help ensure that the system remains responsive and that resources are used efficiently.
4. **Energy efficiency:** CRIU can be used to enable energy-efficient computing in edge computing systems. By migrating running applications to hosts that are currently idle, CRIU can help reduce overall energy consumption and improve the sustainability of the system.

One of the main points of interests is the advent of decentralized protocols for autonomous container management and orchestration. Protocols such as Caravela [6], and Nion Network [7] proposed a decentralized network of

nodes that perform container migrations in an effort to balance resource utilization across the entire network. Nion network secures the decision making process through an efficient and scalable blockchain consensus mechanism. Unlike typical blockchain protocols, the blocks are viewed as states denoting the state of all the containers under management. Adding new blocks to the chain can be considered as the state transition, while the state transition function can be viewed as equivalent to nodes performing planned migrations of containerized applications between peers.

Clearly, such networks are subject to Byzantine behavior and while consensus mechanisms blockchains typically use and tolerate such faults, the applications do not. These faults are traditionally addressed by introducing redundancy where possible and/or backups. In this use-case, redundancy would require more nodes to run the containerized application and serve as readily available backups that take over in case of a fault. Evidently, this does raise concern over the efficiency of the entire network due to additional compute/storage resource utilization. On the other hand, backups only impose additional storage requirements but they have to be frequent to avoid loss of states in-between snapshots.

CRIU can be used to regularly snapshot applications and commit these snapshots to other nodes. In an event of a fault on the host machine(node), the protocol can detect it and restore the application on another node. However, given the experimental nature of CRIU, the exact resource requirements for creating checkpoints and restoring them needs to be well understood. Moreover, studying the telemetry while performing such operations can give a better understanding into the limitations of run-time migrations.

In this paper, we explore the potential benefits of using CRIU in edge computing systems using Docker experimental features that support CRIU [4, 8]. We present an empirical study of the performance impact of CRIU on Docker containers and investigate the feasibility of using CRIU in edge devices. Specifically, we examine the impact of the size of the memory allocation of the host system on the time to checkpoint and restore, as well as the resources consumed during the process execution.

Our results demonstrate the potential benefits of using CRIU in edge computing systems, providing insights into the performance impact of using this tool in these settings. We believe that this research will contribute to the growing body of work exploring the use of checkpointing and migration in edge computing systems and will pave the way for further research into the practical applications of CRIU in this domain.

2 Related Work

CRIU, which stands for “Checkpoint/Restore In Userspace,” is an open-source software tool for performing live migrations of Linux processes. It was first introduced in 2011 by Pavel Emelyanov as a way to checkpoint and restore individual processes in Linux user space, with the goal of enabling faster live migrations for cloud computing workloads. Since then, CRIU has gained popularity in both cloud and edge computing settings, where it allows for efficient and flexible management of distributed applications.

Since its inception the project has matured considerably. One of the earliest studies analyzing the performance of checkpoint and restore was published by Chen Yang [2]. Yang analyzed the time it took to checkpoint and restore a process and unidentified process memory allocation to be the most impactful factor. However, Yang’s study is very limiting as only small processes were used, and no other telemetry such as CPU utilization was analyzed while performing migrations.

Oh, SeungYong, and JongWon Kim experimented the use of CRIU for live migrations of containerized applications [5]. In their study, they analyze the performance of CRIU with a focus on latency analysis in transferring checkpoints between hosts and were less concerned about the resources and time needed to perform these tasks.

A similar study in which CRIU was used to improve the availability of Docker Swarm, a cluster orchestration solution provided by Docker [3]. Their findings report that higher availability can be achieved by periodically checkpointing the states of containers within the swarm and restoring them to provide higher availability. Similarly to the study of Oh, SeungYong, and JongWon Kim, their analysis is focused on the memory use and storage on the network file system (NFS), and did not study the impact on the computing resource utilization while performing CRIU. Moreover, their study is very limiting with very few scenarios and container configurations analyzed.

A more recent study by Adityas et al. analyzed the resource utilization of the host machine while performing checkpoint and restore [9]. In their study, they report the expected memory allocation of the container to have the most impact on the time it takes to perform checkpoint and restore. Interestingly, their study also reports on the CPU consumption while performing these tasks. However, instead of extensively testing various containers, the authors chose a scenario based approach. They identified three main scenarios, namely *one way migration between two hosts one direction at a time*, *two way migrations using one service*, and *two way migrations using three services*.

While the preliminary results reported do offer some insight, the study is very limited to the aforementioned scenarios. The study would benefit a more detailed analysis of multiple services with various memory allocations and a much larger sample size.

Previous research suggests the use of CRIU is ubiquitous in edge and micro-service architectures. However, the lack of an in-depth performance analysis that highlights the limitations is evident. Our study aims to address the gap and pave the way for future use of CRIU for run-time application migrations on the edge.

3 Proposed Approach

Migrating applications between edge devices is required for latency optimization in real-time communication services where minimizing the distance between the user and the service can significantly reduce latency. By migrating containerized services closer to the edge nodes located near the end-users, the improvement in latency can significantly improve the quality of the service and overall user experience.

Another significant scenario is content delivery networks (CDNs) where distributing content across a network of edge devices to deliver it more efficiently to end-users is desired. This is especially important when content is geographically dependent and moving content providers to edge nodes, which are geographically close to the content greatly reduces stress on the central servers.

In Internet of Things (IoT) deployments, migrating computation to edge devices not only brings the compute closer to where the data is located, avoiding central data aggregation and processing but also improves overall usability of currently underutilized resources available on IoT devices.

Migrating containerized applications has also seen considerable interest in mobile edge computing (MEC). MEC brings computational capabilities closer to mobile users by deploying edge servers at base stations or access points. Containers hosting mobile applications or services can be migrated to MEC nodes based on user location or network conditions. This enhances application performance, reduces network congestion, and enables new edge-enabled services such as augmented reality or context-aware applications.

Recently, edge-based AI inference has adopted the edge paradigm where edge devices increasingly utilize AI models for tasks like image recognition, natural language processing, or predictive maintenance. By migrating containers containing AI inference engines or pre-trained models to edge nodes,

you can perform inference locally, reducing latency and privacy concerns associated with sending data to centralized servers for processing.

Nion Network is a protocol for decentralized containerized application migration that covers all of the aforementioned use cases and the main use case used for this research. The aim of the protocol is for the network to balance resource utilization across the network by performing container migrations from nodes where compute resources are heavily used to nodes where compute is available. The protocol includes a decentralized orchestrator akin to Kubernetes but guarantees that autonomous migrations are secured by consensus. Nion requires container migrations to perform as fast as possible, hence CRIU is used to extract the container's context and migrate it to another node. To perform migrations of containers at runtime, two options are available. The common method provided via docker API is to pause the container, export it to an image, transfer the image to the destination host, and restore the container from the image. The second option, which assumes the base image used to create the initial container is available on both nodes, is to use CRIU to pause the running container, extract the context only, transfer the context to the destination, create a new container from the base image and inject the context and run it. The following Bash script assumes CRIU and Docker are installed on the host. The first script takes care of exporting and compressing the container or CRIU state depending on the `-c` parameter denoting the use of CRIU. The second parameter is the previously extracted payload, and the third is the base image used to create the container.

```

1  #!/bin/bash
2  if [ $1 == "-c" ]; then
3  CONTAINER=$2
4  docker checkpoint create --checkpoint-dir='/tmp' $CONTAINER $CONTAINER
5  tar -C /tmp -cf /tmp/${CONTAINER}.tar $CONTAINER
6  rm -R /tmp/${CONTAINER}
7  else
8  CONTAINER=$1
9  docker stop $CONTAINER
10 COMMIT='docker commit $CONTAINER | cut -c 8-'
11 docker save $COMMIT > "/tmp/${CONTAINER}.tar"
12 fi

```

The second script can be used to restore a container using the compressed payload from the first script and provides the `-c` parameter for enabling or disabling the use of CRIU. The second parameter is the compressed state or container image obtained from running the first script.

```

1  #!/bin/bash
2  if [ $1 == "-c" ]; then

```

```

3  MIGRATED_CONTAINER=$2
4  IMAGE=$3
5  CONTAINER='docker create $IMAGE 2> /dev/null | tail -n 1'
6  tar -xf "/tmp/${MIGRATED_CONTAINER}.tar" -C \
7      "/var/lib/docker/containers/${CONTAINER}/checkpoints/"
8  docker start --checkpoint=$MIGRATED_CONTAINER $CONTAINER
9  echo $CONTAINER
10 else
11  MIGRATED=$1
12  CONTAINER='docker load -i "/tmp/${MIGRATED}.tar" | sed `s:/:\n/g` | tail -n 1'
13  docker run -d $CONTAINER
14 fi

```

Figure 1 shows the migrations on the decentralized edge network built using Nion protocol over a span of 3 hours. Nodes in the network were running the decentralized orchestrator and performed migrations every block. The block time was set to 10 seconds, and limited to one migration per block (not a protocol level limitation). The containers were running a Rust application, which periodically reported their liveness status to a web service to verify that the migration successfully restored the state of the container. The memory allocation of each node was limited to 128 MB with 1 CPU to

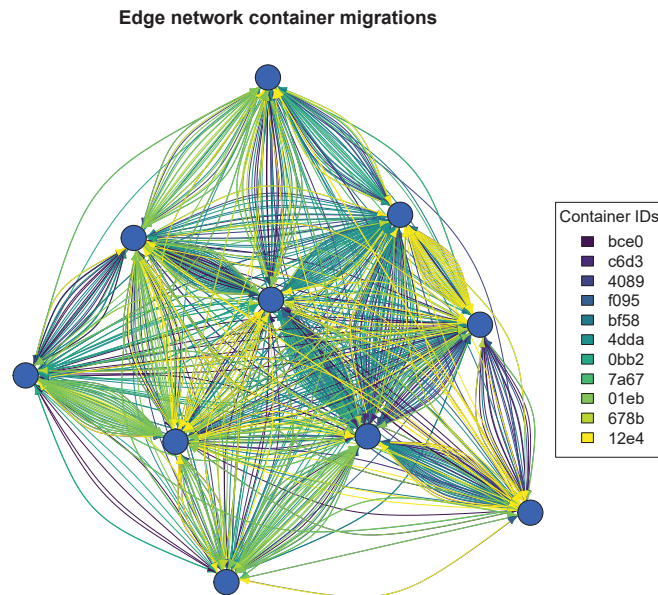


Figure 1 Real-time container migrations on a P2P edge network of nodes. Containers were tracked by their network-wide Id (hash), and migrated between nodes with the goal of balancing workload across the network.

Table 1

Phase	Min(s)	Median(s)	Mean(s)	Max(s)
Resume	1.79	2.13	2.242	9.57
Save	1.41	1.73	1.771	3.93
Transfer	0.032	0.058	0.05787	0.275

mimic typical resource constraints of edge devices, while the CRIU snapshot size was 16M. Average migration times varied significantly due to other network operations such as voting, signature verification, consensus protocol execution, blockchain validation, and maintaining the P2P network executing the Nion protocol. Table 1 offers some insight into the performance of CRIU on edge devices. We observe that transferring snapshots between nodes to be the least impactful, which is expected given the simulation was run on a HPC cluster running docker swarm with 1 Gbps Ethernet links between nodes. We reported no failed checkpoints or restores, which reinforces the premise of CRIU being robust.

4 Experimental Results

When CRIU is used to checkpoint a running application, it performs the following steps:

1. Image creation: CRIU creates an image of the process that needs to be checkpointed. This image contains information about the process's memory, open files, network connections, and other relevant information. The image is created in a checkpoint file.
2. Freezing: Once the image is created, CRIU freezes the process. Freezing is the process of pausing the execution of the process so that its state does not change while the checkpoint is being created.
3. Image dumping: CRIU dumps the image of the process into the checkpoint file. This process involves saving the state of the process's memory, CPU registers, file descriptors, network connections, and other relevant information.
4. Thawing: After the image has been dumped, CRIU unfreezes the process and allows it to resume execution.

The checkpoint file created by CRIU contains all the information necessary to restore the process's state at a later time. When the process needs to be restored, CRIU reads the checkpoint file and recreates the process from the saved state. This process involves creating a new process, restoring the

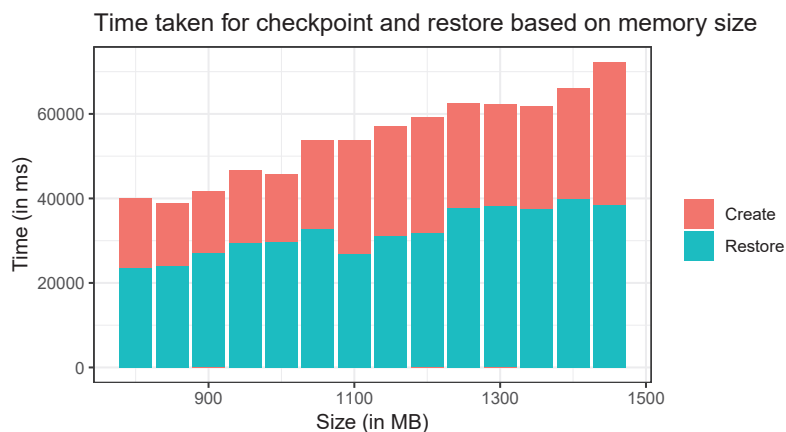


Figure 2 CRIU checkpoint and restore timings relative to the amount of memory used by the Docker container.

process's memory, CPU registers, file descriptors, network connections, and other relevant information, and then resuming the process's execution.

The ability to store and restore the state of a running application is possible due to the design of the Linux kernel. Linux uses a virtual memory system to manage memory, which allows processes to have their own virtual address space. This means that each process has its own view of memory, and the kernel can manage the physical memory separately for each process. When a process is checkpointed, its virtual memory state is saved, including the contents of the process's memory and its CPU registers. This allows the process to be restored to the same state later, even if it is running on a different host or at a different time.

Memory is obviously the main factor that impacts the performance of CRIU. To get a better understanding of the relationship between the size of virtual memory applications require and CRIU performance, a program written in Rust was deployed as a Docker container, the program allocates a given amount of memory and runs using the Ubuntu 20.04 as the base image. After the initial allocation, an external process checkpoints the container and restores it after while measuring the timings for individual tasks. Figure 2 illustrates the timings (checkpoint and restore) observed for various memory allocation sizes.

We can observe that the time it takes to perform a checkpoint is proportional to the amount of memory the container had allocated. While this is not surprising, it does set some boundaries on how checkpointing can be used.

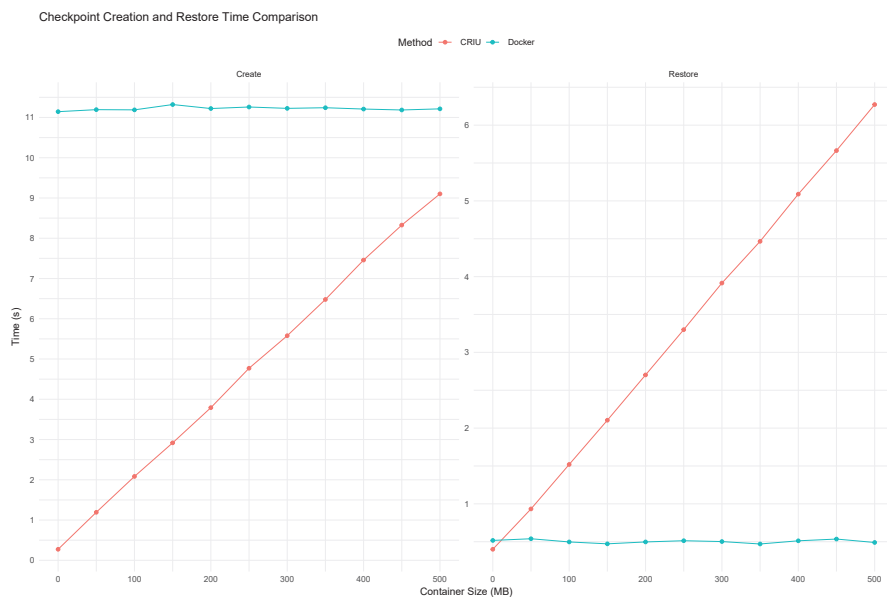


Figure 3 Comparison between CRIU and Docker commit and restore times.

It's important to note that Docker provides a way to migrate a running container. However, these migrations are not quite the same since the program state will not be fully restored. Migrating a Docker container can be done by pausing the running container followed by a commit, which will save the changes in the file system into an image. The image can then be restored on any host without the runtime context. Regardless of its limitation it is still useful to compare the two methods since one is a subset of the other. Figure 3 shows the comparison between CRIU and Docker commit on both creating the checkpoint and restoring it. We observe that without the ability to checkpoint working memory and processes, the commit, and restore from Docker are constant regardless of container size, while CRIU scales have already been observed in Figure 2.

Moreover, the analysis shows that CPU utilization does not seem to scale with the size of virtual memory allocated during the checkpoint and restore phases. This suggests that CPU utilization may not be a limiting factor in checkpointing applications, and the trade-off between checkpointing and performance may be acceptable for stateless applications with limited resources available on edge devices.

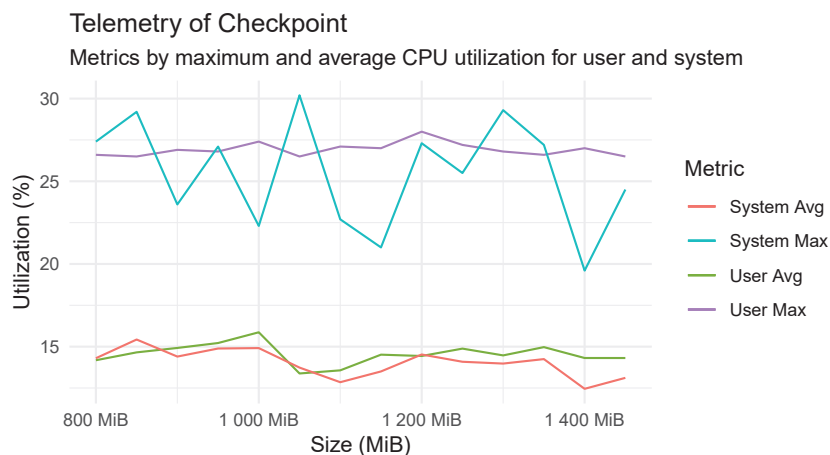


Figure 4 CRIU checkpoint and restore timings relative to the amount of memory used by the Docker container.

Note that the container has to be paused in order to dump the pages of virtual memory and maintain consistency. Checkpoints of applications, which require high availability and constant up-time (i.e. servers) do not seem feasible. However, micro-services and stateless applications seem reasonable. The absence of a global state in applications greatly reduces the memory used, making CRIU a viable approach for scalability and fault tolerance.

However, edge computing solutions have to consider limited resources available on edge devices. While the time it takes to perform a checkpoint and restore may be a trade-off worth taking for stateless applications, the CPU utilization on the host device may be a limiting factor.

To gain some insight into the CPU utilization of the host dockerd service and CRIU we monitor system telemetry when performing checkpoint and restore tasks. Figure 4 illustrates the CPU utilization in per cent both of the user running the checkpoint as well as the overall system. We can observe that while the memory allocation of the application directly impact the time to perform these tasks, the CPU utilization does not seem to scale with the size of the virtual memory allocated. Moreover, the same can be concluded for the restore phase observed in Figure 5.

Overall, the analysis highlights the potential benefits and limitations of using CRIU for checkpointing applications and provides insights into the trade-offs between performance, memory usage, and fault tolerance.

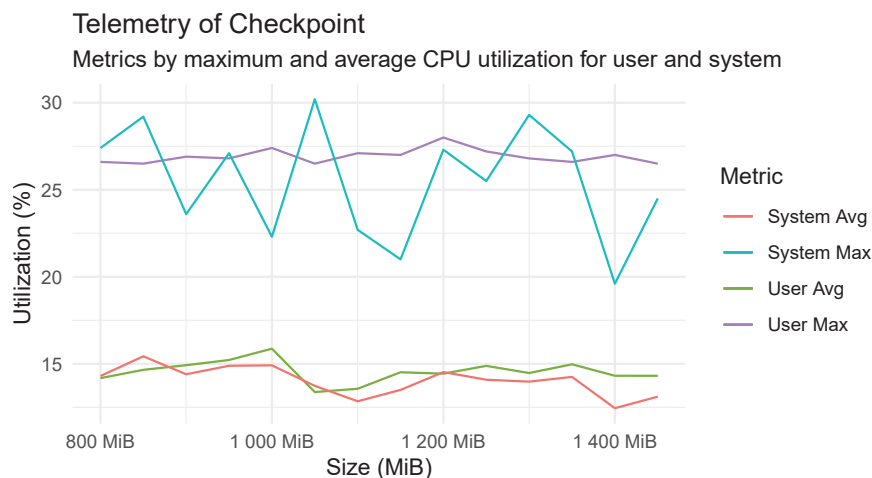


Figure 5 CRIU checkpoint and restore timings relative to the amount of memory used by the Docker container.

5 Conclusion

In conclusion, we have investigated the capabilities and limitations of the CRIU tool for checkpointing and restoring containers in the context of edge computing. Our findings demonstrate that CRIU, albeit experimental, is reliable in migrating docker containers, provided the base image is available on the destination host. However, the size of the virtual memory allocated by the application greatly impacts the time required to perform checkpoint and restore operations. Thus, edge computing solutions should consider the trade-off between checkpointing frequency and application performance. Additionally, while the CPU utilization does not seem to scale with the size of the virtual memory allocated, it remains a limiting factor in the deployment of checkpointing on edge devices with limited resources.

Future work can explore techniques to reduce the memory footprint of the container, enabling more frequent checkpoints without affecting the application's performance. Furthermore, it would be interesting to investigate the use of CRIU for more complex statefull applications and the integration of CRIU with other container orchestration tools. Overall, our study contributes to the understanding of the capabilities and limitations of CRIU in the context of edge computing and provides insights for the deployment of checkpointing solutions in resource-constrained environments.

Acknowledgement

The authors gratefully acknowledge the European Commission for funding the InnoRenew project (Grant Agreement #739574) under the Horizon2020 Widespread-Teaming program and the Republic of Slovenia (Investment funding of the Republic of Slovenia and the European Regional Development Fund). This project has received funding from the European Union's Horizon 2020 research and innovation program through the NGI TRUSTCHAIN program under cascade funding agreement No. 101093274

References

- [1] Fabio Andrijauskas, Igor Sfiligoi, Diego Davila, Aashay Arora, Jonathan Guiang, Brian Bockelman, Greg Thain, and Frank Wurthwein. Criu-checkpoint restore in userspace for computational simulations and scientific applications. *arXiv preprint arXiv:2402.05244*, 2024.
- [2] Yang Chen. Checkpoint and restore of micro-service in docker containers. In *2015 3rd International Conference on Mechatronics and Industrial Informatics (ICMII 2015)*, pages 915–918. Atlantis Press, 2015.
- [3] Cheng-Hao Huang and Che-Rung Lee. Enhancing the availability of docker swarm using checkpoint-and-restore. In *2017 14th International Symposium on Pervasive Systems, Algorithms and Networks & 2017 11th International Conference on Frontier of Computer Science and Technology & 2017 Third International Symposium of Creative Computing (ISPAN-FCST-ISCC)*, pages 357–362. IEEE, 2017.
- [4] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.
- [5] SeungYong Oh and JongWon Kim. Stateful container migration employing checkpoint-based restoration for orchestrated container clusters. In *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 25–30. IEEE, 2018.
- [6] André Pires, José Simão, and Luís Veiga. Distributed and decentralized orchestration of containers on edge clouds. *Journal of Grid Computing*, 19(3):1–20, 2021.
- [7] Aleksandar Tošić, Jernej Vičič, Michael David Burnard, and Michael Mrissa. A blockchain-based edge computing architecture for the internet of things. 2022.
- [8] Ranjan Sarpangala Venkatesh, Till Smejkal, Dejan S. Milojicic, and Ada Gavrilovska. Fast in-memory criu for docker containers. In *Proceedings*

of the *International Symposium on Memory Systems*, MEMSYS '19, page 53–65, New York, NY, USA, 2019. Association for Computing Machinery.

- [9] Adityas Widjajarto, Deden Witarsyah Jacob, and Muharman Lubis. Live migration using checkpoint and restore in userspace (criu): Usage analysis of network, memory and cpu. *Bulletin of Electrical Engineering and Informatics*, 10(2):837–847, 2021.

Biography



Aleksandar Tošić is Assistant Professor at the University of Primorska, and Researcher at InnoRenew CoE. His main research interests are distributed systems, privacy and security, sensors and distributed ledger technologies.