Master Thesis Proposal

Time-aware Container-based Virtualization

Stefan Walser, 11809605

Supervisor: Privatdoz. Dipl.-Ing. Dr.techn. Wilfried Steiner
Assistant: Dr. Silviu S. Craciunas
Assistant: Jan Ruh, M.Sc.

April 20, 2023

1 Motivation and Introduction

Since the introduction of containers to Linux in 2008 [1] they gained a lot of popularity in cloud computing. Their scalability, minimal image size, and runtime performance make them suitable for the often used microservice architecture. This architecture found more and more use during the last decade due to the possibility of independent development and deployment of the different services applications consist of [2].

Previously, cloud applications, mostly implemented in a monolithic fashion, were often virtualized with the use of a hypervisor [3]. A Hypervisor is a software that abstracts the hardware of the physical machine. Multiple virtual machines (VM) can then run on top of the abstracted hardware [4]. The ability to have multiple VMs on one physical machine has the benefit of better resource utilization, which leads to lower costs and a reduced power usage [5] and is the reason for its popularity in the area of cloud computing [6]. However, as it is the hardware which is abstracted, it is necessary to have a full-fledged operating system running inside each VM. This results in greater image sizes and longer boot times [7]. A VM image is an executable file which contains the software that runs inside the virtual machine [3].
Container-based virtualization on the other hand does not require a hypervisor. The virtualization is realized with a combination of operating system kernel features. This has the benefit of smaller images and fast boot times of a few milliseconds [5]. In Linux, containers are realized with the use of the two kernel features called *namespaces* and *control groups (cgroups)*. Namespaces isolate and partition the resources of the machine in such a way that different processes see only a part of the resources. Cgroups monitor and limit the access of processes to resources [8]. However, as the separation of the containers is not realized by hardware, their influence on each other is substantial. This has to be considered, especially when implementing RT systems [9].

As Struhár et al. describe in [10], the ability to co-locate multiple containers on a single machine makes containers attractive to mixed-criticality system engineers in the areas of industrial automation, aviation, and automotive for two reasons. Firstly, because different computational resources can be consolidated on fewer devices and secondly, because it enables interruption-free hardware

and software maintenance.
However, these mixed-criticality applications, when implemented in a Linux environment, can not satisfy hard RT and safety-critical constraints. This is because of the complexity of the Linux kernel, which makes it impossible to guarantee a WCET with the use of classical static timing analysis [11].
Moga et al. examined in [12] the benefit of using containers in industrial automation systems. These are systems that control, monitor and supervise for example factories or plant processes. More and more devices are needed to scale the functionality and the size of these systems. However, this comes with an increase in factory footprint, i.e. the physical size of the factory, and higher maintenance costs. In such cases, containers can consolidate multiple systems on the same hardware.
Network operators also start to replace physical network appliances with containerized software functions. This is called Network Function Virtualization. It makes it easier to satisfy the consumers needs, when it comes to high-reliability, high-availability and high-performance. Containers can be easily duplicated which is not only necessary at high-load times, but also to mitigate errors in other containers [13].

However, besides these benefits there is a critical point in these applications: a need for RT guarantees.
As previously mentioned, there are multiple container instances running on the same physical machines which can influence each other. Therefore, it is mandatory to make sure, that this influence is kept to a minimum and each container gets the share of the resources it needs. If at one point it is not possible anymore to fulfill the requirements of a container, the first counter-measure is to throttle or stop non-RT containers. If this is not sufficient, it is necessary to relocate one or more containers to another physical node. This relocation, together with deploying the containers to nodes initially, is the task of an orchestration tool, which is almost mandatory for a system that uses container-based virtualization with a large number of nodes [14].

2 Aim of the Thesis

The aim of this thesis is to extend the work of Struhar et al. [9, 14] in the following points:

- We enhance the SMT-solver based approach to initially orchestrate the containers to the individual nodes with heuristics. The heuristics enable optimizing the orchestration e.g. to minimize energy consumption or fulfill a specific time-utility for a subset of tasks.
  Time-utility functions describe the value of a result over time [15].
- We define metrics to extend an existing runtime monitor to report on the performance of a node's containers concerning energy consumption and time-utility. The information is propagated to the orchestrator to migrate containers between nodes if the power consumption or time-utility requirements cannot be met.
- We evaluate the framework not only in terms of response-time of the tasks, but also in terms of lateness, number of deadline misses, power consumption and time-utility.

These additions to the framework enable the definition of power consumption and time utility requirements so that we can find optimized allocations during design time. The extensions to the runtime monitor and orchestrator aid self-configuration, self-diagnose, and self-adaption of cloud-to-edge systems. The consideration of power consumption during scheduling will make it possible to use the framework in environments where power is a limited resource.

3 Methodological Approach

To successfully conduct this masters thesis, the following steps will be necessary:

1. **Literature Review:** The first step is to get a deep understanding of how containerization for RT systems works, what can be done to orchestrate the containers in such a way that their requirements are met and which systems have already been developed to achieve RT orchestration. Furthermore, a heuristic to solve the optimization problem regarding the power consumption and time-utility needs to be found. To find a technique to monitor the power consumption of the nodes is also part of the review.
2. **Evaluate Heuristic:** To make sure the heuristic is suitable, as a second step, before making the system more complex with other metrics, the heuristic will be evaluated.
3. **Theoretic Contribution** The goal of this step will be to extend the metrics of the offline and online phase. This includes power-consumption and the time-utility functions.
4. **Implement Metrics** In this step the new metrics will be implemented into the controllers.
5. **Evaluation:** As a final step the framework will be evaluated to gain insight into the performance with respect to different container-level and OS-level metrics, e.g. response time, lateness or number of deadline misses and also power consumption and time-utility.

4 State of the Art

There has already been research in the direction of enabling RT behaviour in containers. The following three approaches are mainly used throughout literature [10]:

PREEMPT_RT Patch The real-time patch for the Linux kernel increases its preemptible code. This includes for example critical sections and interrupt handlers [16].

Real-time Co-Kernel The additional real-time micro-kernel, that runs parallel to the Linux kernel, takes over all the time-critical tasks. This includes interrupt handling and scheduling RT threads. Whenever this kernel is idle, the standard Linux kernel is executed. This solution has the drawback, compared to the PREEMPT_RT ptach, that additional APIs, tools and libraries are necessary to develop applications [17].

Hierarchical Scheduling This approach schedules on two levels. On the first level an Earliest Deadline First (EDF) scheduler selects the container to be scheduled. On the second level a fixed priority scheduler schedules the tasks inside the container. This variant is the one that is used by Struhár et al. in [9, 14].

However, these enhancements to the kernel are not enough to fully utilize the potential of containerization. An orchestration tool is necessary to automatically deploy, re-dimension and relocate containers. In order for this tool to be usable in a RT environment, it must monitor the nodes in the system and the containers running on them.

Fiori et al. presented in [18] an adaption of the orchestration framework Kubernetes that is able to orchestrate containers in such a way that timeliness is guaranteed. On every node on the system they used the Hierarchical Constant Bandwidth Server scheduler ([8]) together with the PREEMPT_RT patch. A "manifest" file in YAML format can then be used to specify the Pods. A Pod is a concept of Kubernetes and consists of multiple containers (in RT-Kubernetes only one), network connections

and storage. In order for Kubernetes to be able to schedule the Pod to a node that is capable of executing the contained application, the manifest file must contain additional RT information. This is, apart from the amount of CPU cores (rt_cpu) the container should run on, the runtime Q and the period P of the container. The scheduler then chooses one of the nodes to schedule the Pod on where it is guaranteed that every P execution time units the container can run for a time Q on rt_cpu cores. This so called admission test is realized via a utilization based calculation and can also guarantee hard RT behaviour.
Compared to the approach of Fiori et al. which is not able to consider other factors than RT requirements for the orchestration, this thesis will provide a framework which is able to consider the power consumption of the nodes and the time-utility functions of the tasks too.

Struhár et al. developed a similar approach in [14]. However, they implemented their own orchestration framework. It consists of an online and an offline phase. The online phase monitors the system on three different layers to make sure that it adheres to the timing requirements. The offline-phase calculates an initial container placement. This initial calculation uses various different equations which describe the systems resources and the requirements of the tasks. An SMT solver is then used to find an assignment that fulfills all the needs that were specified by the equations. After all containers started, there is a container-level monitor that is able to free CPU resources if the container it is monitoring doesn't fully utilize its assigned quota. The node-level monitor on the other hand has information on the usage of all the containers running on its node and therefore has the ability to increase the CPU-quota of a container when its current quota is not sufficient. If the node-level monitor detects that it is not able to fulfill the needs of all its assigned containers, it marks a container for relocation and the cluster-level monitor then searches for another node and migrates the container to it.
This Thesis will extend the work of Struhár et al. by considering power consumption and time-utility functions of the tasks during the offline and online phase.

5 Relevance to the Curricula of Computer Engineering

For this thesis knowledge in the area of container-based virtualization, scheduling, especially in a real-time environment, and operating systems is necessary. Additionally, basic skills in the field of algebra are required too.
These topics were covered in the following courses of the Computer Engineering curricula:

- 182.713 VO Real-Time Systems
- 184.237 VO Distributed Systems
- 182.711 VO Operating Systems
- 182.709 UE Operating Systems
- 104.265 VO Algebra and discrete mathematics

References

[1] Notes from a container. `https://lwn.net/Articles/256389/`. Accessed: 16.03.2023.

[2] Genc Mazlami, Jürgen Cito, and Philipp Leitner. Extraction of microservices from monolithic software architectures. *2017 IEEE International Conference on Web Services (ICWS)*, pages 524–531, 2017.

[3] Claus Pahl. Containerization and the paas cloud. *IEEE Cloud Computing*, 2(3):24–31, 2015.

[4] Roberto Morabito, Jimmy Kjällman, and Miika Komu. Hypervisors vs. lightweight virtualization: A performance comparison. In *2015 IEEE International Conference on Cloud Engineering*, pages 386–393, 2015.

[5] Michael Eder. Hypervisor-vs. container-based virtualization. *Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)*, 1, 2016.

[6] Nancy Jain and Sakshi Choudhary. Overview of virtualization in cloud computing. In *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, pages 1–4, 2016.

[7] Anuj Kumar Yadav and ML Garg. Docker containers versus virtual machine-based virtualization. In *Emerging Technologies in Data Mining and Information Security: Proceedings of IEMIS 2018, Volume 3*, pages 141–150. Springer, 2019.

[8] Luca Abeni, Alessio Balsini, and Tommaso Cucinotta. Container-based real-time scheduling in the linux kernel. *SIGBED Rev.*, 16:33–38, 2019.

[9] Václav Struhár, Silviu S. Craciunas, Mohammad Ashjaei, Moris Behnam, and Alessandro V. Papadopoulos. React: Enabling real-time container orchestration. In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, 2021.

[10] Václav Struhár, Moris Behnam, Mohammad Ashjaei, and Alessandro Vittorio Papadopoulos. Real-time containers: A survey. In *Workshop on Fog Computing and the Internet of Things*, 2020.

[11] Federico Reghenzani, Giuseppe Massari, and William Fornaciari. The real-time linux kernel: A survey on preempt_rt. *ACM Comput. Surv.*, 52(1), feb 2019.

[12] Alexandru Moga, Thanikesavan Sivanthi, and Carsten Franke. Os-level virtualization for industrial automation systems: are we there yet? *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016.

[13] Tommaso Cucinotta, Luca Abeni, Mauro Marinoni, Alessio Balsini, and Carlo Vitucci. Virtual network functions as real-time containers in private clouds. 07 2018.

[14] Václav Struhár, Silviu Craciunas, Mohammad Ashjaei, Moris Behnam, and Alessandro Papadopoulos. Hierarchical resource orchestration framework for real-time containers. *ACM Transactions on Embedded Computer Systems*, 7(1).

[15] E.D. Jensen. A timeliness model for asychronous decentralized computer systems. In *Proceedings ISAD 93: International Symposium on Autonomous Decentralized Systems*, pages 173–182, 1993.

[16] A realtime preemption overview. `https://lwn.net/Articles/146861/`. Accessed: 24.03.2023.

[17] Timur Tasci, Jan Melcher, and Alexander Verl. A container-based architecture for real-time control applications. In *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, pages 1–9, 2018.

[18] Stefano Fiori, Luca Abeni, and Tommaso Cucinotta. Rt-kubernetes: containerized real-time cloud computing. pages 36–39, 04 2022.