RT-SCALER: Adaptive Resource Allocation Framework for Real-Time Containers

Václav Struhár[1], Silviu S. Craciunas[2], Mohammad Ashjaei[1], Moris Behnam[1], and Alessandro V. Papadopoulos[1]
[1]Mälardalen University, Västerås, Sweden; [2]TTTech Computertechnik AG, Vienna, Austria

***Abstract*—Container-based virtualization has emerged as an advantageous deployment model in fog computing platforms since it enables the seamless co-location of applications in a heterogeneous environment with minimal overhead. For some application domains requiring a certain degree of predictability in the time domain (e.g., industrial automation), the adoption of container-based virtualization is not straightforward since the technology is not built to support real-time properties.**

In this paper, we propose RT-SCALER, which is a framework for adaptive resource allocation and dimensioning for real-time containers. RT-SCALER dynamically adapts the resource reservation of real-time enabled containers in order to improve the temporal predictability of the real-time applications running within the containers. We discuss the high-level orchestration approach, relating the different control levels, and give some practical insights into node-level container adaptation.

I. Introduction

Container-based virtualization has emerged as a suitable deployment model in fog computing [1] and edge platforms [2], [3], [4], as it provides near-native performance with low memory footprints and rapid start-up times. Additionally, containers provide portability, ensuring that applications will work the same way regardless of the environment where they are deployed. Moreover, containers eliminate the additional overhead of dedicated virtualization layers and, thus, use the host's available resources more efficiently. The properties mentioned above make container-based virtualization a suitable technology for hosting applications in heterogeneous distributed environments, such as fog and cloud computing platforms. However, using containers in domains imposing safety and real-time requirements (e.g., industrial automation), requires that containers are spatially and temporally isolated and can offer some form of timing predictability for the underlying applications. While spatial isolation is a fundamental property of containers, support for real-time is still under ongoing research.

In general, real-time properties relate to the ability of applications to produce not only correct logical results but also to uphold temporal limits (deadlines) when computing the results [5]. Such temporal requirements are challenging, especially in heterogeneous environments with a dynamically changing number of containers with unpredictable workloads and access patterns to shared resources. However, only limited attempts have been made to enable real-time behavior in container-based virtualization within this research area. For instance, the Hierarchical Constant Bandwidth Server (HCBS) solution by Abeni et al. [6] provides temporal isolation of containers that share the same physical host. The HCBS consists of two-level schedulers, the global scheduling policy for the containers, while the second level is based on fixed-priority scheduling for software tasks.

Nevertheless, at runtime, container-based virtualization is prone to resource interference. As the containers share the operating system kernel of the host, the performance interference, that is, the performance isolation problem will occur between the containers due to the resource competition [7]. Resource interference may influence the execution times of the containerized applications and introduce timing unpredictability. Within this context, the HCBS solution [6] specifies CPU time reservation for RT containers in the form of a certain percentage of the CPU bandwidth over a specified period. The reservation is commonly done via reserving a CPU *budget* over the *period*. Choosing the correct amount of the budget and period is a non-trivial task, as it directly affects the timing properties of the containers. The reservation problem becomes even more challenging in heterogeneous and dynamic systems due to (i) the worst-case execution time (WCET) on the specific platform being unknown beforehand, (ii) due to the unpredictable performance interference originating in other co-located containers, and (iii) due to possible dynamic workload changes in the RT containers. Therefore, it is not sufficient to compute the resource reservation in terms of the budget and period at design time (offline phase), but it is also necessary to adapt it at runtime to improve the utilization of resources and the time-predictability of services (online phase).

In order to achieve the goal of time predictability of container-based virtualization, we propose an orchestration framework named RT-SCALER in this paper that considers both the offline and online aspects of the adaptation problem for real-time containers. Besides an offline phase, in which the container dimensioning can be done based on an static system model without considering runtime overhead, we propose an online phase that is able to respond to changes in the performance of real-time tasks as well as to changes in the required workload (e.g., when new applications or containers join the system), in order to both preserve the timeliness of applications and to utilize the available resources more efficiently. We describe the overall design of RT-SCALER along with its components and highlight the hierarchical nature of the control problem for runtime container adaptation (Sec. II). Additionally, we discuss some practical insights and experimentally show the benefits of controlling the resources

This work has been performed with the support from the Swedish Knowledge Foundation (KKS) under the SACSys project (#20190021), and from the Swedish Research Council (VR), under the PSI project (#2020-05094).

during runtime at the local node level (Sec. III) and draw some conclusions in Sec. IV.

The nature of container-based virtualization provides a ground for resource adaptation. Due to rapid start-up times, it is easy to horizontally scale up the number of containers and balance the workload between them as shown in [8]. Additionally, it is simple to vertically scale the container's resources via cgroups. It is shown in [9] where authors scale container resources based on CPU utilization of containers.

II. Container orchestration

We present the high-level idea of RT-SCALER, which is a general orchestration framework for static and dynamic allocation and dimensioning of real-time containers. Real-time containers supplement the spatial isolation properties of containers with real-time capabilities relating to temporal isolation and deadline fulfillment. Adding real-time properties to containers has been addressed in [6] by introducing a hierarchical scheduling patch for Linux-based systems.

The main aim of our container orchestration, called *RT-SCALER*, is to manage the deployment and adaptation of real-time containers in distributed applications featuring heterogeneous computing nodes such that individual real-time task requirements are met. These requirements are not only related to real-time behavior but also resource usages such as memory, I/O, and disk requirements and non-functional requirements such as fault-tolerance, power consumption, or resource efficiency.

The input to our RT-SCALER orchestrator is a set of real-time tasks and containers. The containers can include either self-suspending real-time tasks with implicit (or constrained) deadlines, in which case they are labeled as real-time (RT) containers or non-real-time tasks, in which case we talk about best-effort (BE) containers. Real-time tasks are additionally defined using a worst-case execution time (WCET) and a period specifying an upper bound on the computation of the task in each period and the rate at which the task is activated. Both RT and BE containers can coexist on the same core, but they are spatially and temporally isolated. The real-time tasks are pre-allocated to containers, but the containers are not pre-allocated to computing nodes (and cores), although they can have a certain affinity set constraining the set of nodes to which they can be allocated to. As defined in [10], each RT container π_k has additionally an RT interface consisting of (P_k, Q_k) where Q_k is the CPU quota within an interval (period) P_k, defining that the container π_k cannot use more than P_k time units over an interval of time of Q_k time units.

We envision our RT-SCALER orchestrator to consist of two phases, an offline and an online one.

A. Offline phase

Given a set of containers and real-time applications as defined above, in the offline phase, RT-SCALER decides where to place the containers such that the real-time requirements of tasks are fulfilled. This decision will be based on two steps.

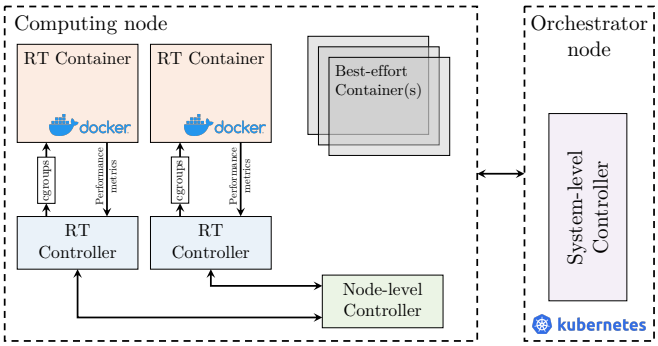

Fig. 1. Overview of the system.

The first step is to calculate a set of ideal RT interfaces for the RT containers, i.e., compute (P_k, Q_k) for every RT container π_k. This is similar to the non-trivial server design problem [11], [12]. In this case, the server design problem is to compute the optimal budget and period of all containers given the set of real-time tasks and their assignment to containers. There are several (computationally intensive) methods to compute the optimal server parameters (e.g. [13], [14]) for both fixed- and dynamic priority schedulers, which rely on a worst-case schedulability analysis of the tasks within the server. Since we know the underlying scheduling mechanism (fixed priority) used to dispatch tasks within a container, we can employ a response time analysis (e.g. [15]) under the worst-case service pattern assumption for the given RT interface to compute the optimal RT interface (P_k, Q_k).

After solving the server design, the allocation of containers to cores becomes an optimization problem similar to the bin-packing problem, and thus NP-hard. There are, however, efficient offline heuristics that can offer near-optimal solutions in a reasonable time even for larger problem sizes [16].

Finally, BE containers need to be assigned, and this can be done either in a random or more balanced fashion. A balanced assignment of BE containers could take into account, e.g., the number of BE tasks in the container and the remaining CPU bandwidth of each node. Once assigned, the remaining bandwidth on each node can be distributed (uniformly or non-uniformly) to the respective BE containers.

The offline phase is not sufficient to guarantee the desired real-time behavior of applications at runtime. It is too complex (and unrealistic) to build an exact model of the underlying system and the runtime interactions between the containers in order to be able to rely solely on the resulting offline dimensioning of containers. Runtime effects may lead to a variance in the temporal behavior since the temporal isolation is not perfect and influenced by runtime artifacts (e.g., cache effects, interrupts) and system overhead. Hence, the ideal server dimensioning done at the offline stage may not lead to the desired behavior in practical deployments. Additionally, new applications or containers can be released in the system, requiring a runtime adaptation of existing containers or assignment of new containers to the available (and possible) cores.

B. Online phase

In the online phase, we envision two components interacting with each other to be able to respond to unforeseen changes in the temporal behavior of runtime tasks. One component is online monitoring of the real-time and health aspects of applications. This aspect is described along with the general framework for deploying and implementing an online container orchestrator module in [10]. In [10], the authors introduced Container Level Metrics (CLM) to capture and continuously evaluate: (i) the number of *deadline misses*, (ii) the *lateness*, and (iii) the response-time of real-time tasks. Additionally, we also monitor Operating System-Level Metrics (OSLM) that give us a picture of the health of the underlying system and the containers. In [10] the authors give special attention to the system overhead, which can affect the temporal isolation property and system utilization which can be used to optimize the response time of tasks as well as to detect overload scenarios or starvation in BE containers.

At runtime, there are several actions that we are able to take based on the CLM and OSLM measurements. The most straightforward parameter to change is the container budget. For example, when detecting a deadline miss of a task, we can increase the budget of the corresponding container, thus giving the tasks within more CPU bandwidth to resume their correct behavior. Naturally, the decision of when and how much to increase the budget is non-trivial as it depends on multiple aspects like the overall system utilization, the effects on other RT and non-RT containers, and the implications on the system overhead. We can also change the period of a container, e.g., to let it run more frequently. This also has complex implications on the overall system behavior and may also affect other tasks running in the same container. Additionally, the implications of changing the period at runtime on preempted but not finished tasks need to be considered. A third dimension where we can enact changes is the container allocation. If we detect at runtime that the system is becoming overutilized or that we cannot guarantee the temporal correctness of all RT tasks and containers, the system may move one or multiple containers onto another (less congested) node. Here, the complexity comes from identifying which containers to move and to which node(s) to move them. Additionally, while the first two parameters were (somewhat) more continuous in nature since we have a whole range of possible values to choose from, the reallocation decision is inherently a binary one. Thus, at runtime, we need to be very careful when switching from slightly adjusting container parameters to deciding that one or multiple containers need to be moved.

The node-level view is depicted in Fig 1 where RT- and BE-Containers coexist in a node, and, additionally, there is one RT-Controller per RT container. The RT-Controller is responsible for adapting local container-level parameters. We do not detail the specifics of what type of controller to use since this is ongoing work, but we envision a runtime adaptation based on control theory. While the RT-Controllers here are local to the RT containers (and hence apply changes to local container values), they need to synchronize and orchestrate to node-level and system-level controllers. By having this controller hierarchy, we can ensure that the holistic view of the distributed system is maintained and the correct overall decisions are made. We envision simple but fast controllers that interact locally with the budget of a container (within some predefined bounds) and can react quickly to runtime violations. Moreover, a node-level controller needs to orchestrate between the RT-Controllers, e.g., to modify the allowed bounds for local budget changes and compute correct dimensioning of, e.g., container periods depending on the overall node-level system state. On the next hierarchical level, a centralized controller needs to orchestrate the migration and reallocation of containers to other nodes.

Another aspect of online redimensioning/reallocation is resource and task optimization. Even when no real-time requirements are violated, the RT-Controllers may decide that there are enough free resources in a node to redimension a particular container (e.g., increasing its budget) to reduce the task response times. Alternatively, an RT-Controller may detect that tasks within an RT-Container finish well before their deadlines and decide to reduce the budget in order to, e.g., optimize non-functional properties such as power consumption or give more bandwidth to BE containers.

III. Practical Insights

This section provides a brief insight into the local control of RT-Containers via a simple PID loop to underscore the potential of runtime adaptation in real-time containers.

The underlying container system in our work is based on the HCBS patch[1] by Abeni et al. [6] that provides temporal isolation of containers sharing the same physical host. The patch is consistent with existing real-time analysis (some results show that it is compatible with the Multiple Periodic Resource Model analysis). The patch hierarchically chains two existing scheduling policies (SCHED_DEADLINE and SCHED_FIFO). The SCHED_DEADLINE scheduling policy implements the Constant Bandwidth Server (CBS) algorithm as the root scheduling policy of the scheduling hierarchy. The second level scheduling policy is fixed-priority scheduling policy SCHED_FIFO. The scheduler provides an interface to control the parameters of the scheduling policies via cgroups. In a cgroups virtual file system, 'cpu.rt_runtime_us' serves to control CPU time reservation for each RT container.

We enhance the Linux Kernel with an online RT task monitoring module and an adaptation module for adapting local container-level parameters. The task monitoring module recognizes containerized real-time tasks that run within the context of HCBS patch, and it continuously collects RT-related performance metrics [5], [10]. In our experiments, we consider the response time of the self-suspending periodic task. However, the module also collects other data consistent with CLM metrics defined in [10]. The module timestamps scheduler-related events. The adaptation module aims to continuously

[1] Available at: https://github.com/lucabe72/LinuxPatches/tree/HCBS

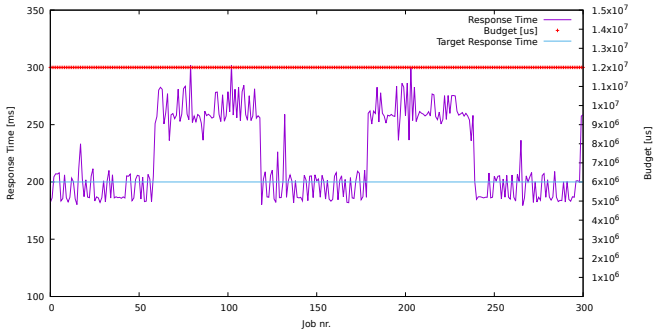

Fig. 2. Response time of a dynamically changing containerized task without online adaptation.

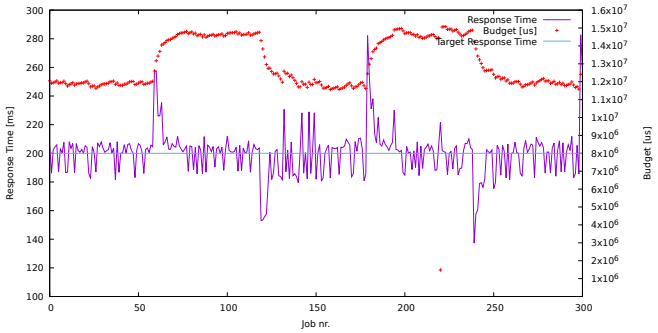

Fig. 3. Response time of a dynamically changing containerized task with online adaptation.

adjust the budget for the real-time containers in a reactive manner in order to keep the real-time performance stable. The adaptation module interacts with the monitoring module to obtain monitoring data (CLM, OSLM) and utilizes a PID controller to adapt the budget of the RT container.

We perform an experiment to demonstrate the adaptation process on the Intel i5 computer with 8GB RAM using Debian Linux (Kernel 5.2.) patched with the Hierarchical Scheduling Patch, and running Docker v20.10.

An experiment showing the feasibility of our idea is depicted in Fig. 2 and Fig. 3. Fig. 2 shows a possible scenario when the response-time of an RT container is affected by unforeseen circumstances. We simulated such a change by changing the workload in the container. At the 60^{th} and 180^{th} job instances, the workload increases by 30%. At the 120^{th} and 240^{th} job instance the workload returns to its original level. The RT budget of the RT container is reserved to fulfill the target response time ($200ms$). However, any workload change or system interference may affect the Quality of Service of the RT container at runtime such that it is not able to deliver demanded performance.

Fig. 3 shows the same scenario that employs RT-SCALER. The monitoring module continuously keeps track of the response times of the containerized tasks. The adaptation module changes the RT budget based on the measured response time. As can be seen, the online adaptation quickly responds to the changing workload and keeps the response time closer to the desired value represented by the blue horizontal line.

IV. Conclusion

We presented RT-SCALER, a container orchestration framework that introduces a two-phased orchestration approach, aiming to preserve RT performance in a multi-container environment. The initial offline phase of the system attempts to compute the theoretically optimal set of RT parameters for the RT containers. This phase is based on the theoretical background of optimal server dimensioning. However, the computed values may not be ideal in real-world heterogeneous systems that may suffer from interference artifacts or workload changes requiring some form of online adaptation. Thus, we introduced an online phase that adapts the RT container values at runtime based on a hierarchical approach. We presented the general RT orchestrator design, proposed the system's architecture, and showed an experiment that demonstrates the feasibility of the idea at the local RT container level.

In future work, we want to investigate different adaptation strategies of real-time containers. For example, the adaptation mechanism could predict workload from previous historical data and proactively dimension the resources. Moreover, we want to address the control challenge of deciding which server parameter to change (including the decision to relocate an RT container to another node) and experimentally evaluate the complex control loop across different hierarchical levels in distributed edge computing applications.

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. MCC*, 2012.
[2] W. A. Hanafy, A. E. Mohamed, and S. A. Salem, "A new infrastructure elasticity control algorithm for containerized cloud," *IEEE Access*, 2019.
[3] C. Dupont, R. Giaffreda, and L. Capra, "Edge computing in iot context: Horizontal and vertical linux container migration," in *Proc. GIoTS*, 2017.
[4] R. Morabito, "Virtualization on internet of things edge devices with container technologies: A performance evaluation," *IEEE Access*, 2017.
[5] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*, 2011.
[6] L. Abeni, A. Balsini, and T. Cucinotta, "Container-based real-time scheduling in the Linux kernel," *SIGBED Rev.*, 2019.
[7] C. Jiqing, "I/O performance optimization analysis of container on cloud platform," in *Proc. ICPICS*, 2020, pp. 84–86.
[8] H. T. Ciptaningtyas, B. J. Santoso, and M. F. Razi, "Resource elasticity controller for docker-based web applications," in *Proc. ICTS*, 2017.
[9] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Autonomic vertical elasticity of docker containers with elasticdocker," in *Proc. CLOUD*, 2017.
[10] V. Struhár, S. S. Craciunas, M. Ashjaei, M. Behnam, and A. V. Papadopoulos, "REACT: enabling real-time container orchestration," in *Proc. ETFA*, 2021.
[11] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Proc. RTSS*. IEEE, 2003.
[12] ——, "Compositional real-time scheduling framework," in *Proc. RTSS*, 2004.
[13] G. Lipari and E. Bini, "Resource partitioning among real-time applications," in *Proc. ECRTS*, 2003.
[14] A. Easwaran, M. Anand, and I. Lee, "Compositional analysis framework using edp resource models," in *Proc. RTSS*, 2007.
[15] L. Almeida and P. Pedreiras, "Scheduling within temporal partitions: Response-time analysis and server design," in *Proc. EMSOFT*. ACM, 2004.
[16] E. G. Coffman, M. R. Garey, and D. S. Johnson, *Approximation Algorithms for Bin Packing: A Survey*, 1996.