Resource Adaptation for Real-Time Containers Considering Quality of Control

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

***Abstract*—Container-based virtualization has become a promising deployment model for industrial applications mainly due to its benefits, such as providing support for co-located applications in heterogeneous environments. However, such facilitation brings challenges, including full temporal isolation among real-time applications and support for time-critical applications. In this paper, we tackle such challenges, in particular when the applications are time-sensitive Control Applications. The literature suggests that flexible timing constraints for Control Applications are beneficial in responding to disturbances and minimizing response deviation. Therefore, we propose a mechanism to support such a runtime adaptation in container-based virtualization. To show the performance of the proposed mechanism, we implement our approach on a Linux-based hierarchical scheduling platform, and we evaluate it for a Control application.**

I. Introduction

The deployment of virtualized applications in heterogeneous environments with minimal overhead and near-native performance is made possible by container-based virtualization, which shows promise in industrial settings [1], [2], [3], [4]. Container-based virtualization is a lightweight virtualization technology that allows applications to run in isolated environments, sharing the host operating system's kernel, enabling efficient resource utilization and easy deployment. Container-based virtualization brings several benefits, including easier application deployment and management, optimized hardware utilization, and increased application flexibility and scalability. This technology also enables the development of microservices-based applications, facilitating agile development, efficient resource utilization, and improved fault isolation. However, despite its advantages, container-based virtualization can pose challenges for applications that require stringent temporal behavior and predictability. This shortcoming can hinder the utilization of containers for time-sensitive controllers which have timing requirements.

Industrial domains often leverage controllers to control the behavior of various systems, such as industrial robots and vehicles. Traditionally, controllers are designed as digital controllers with fixed timing behavior (e.g., fixed periods and deadlines for the control) [5]. Several works [6], [7] show that flexible timing constraints are beneficial in responding to disturbances and minimizing response deviations. Moreover, flexibility can lead to better resource utilization, both in computation and communication. Similarly, in the control community, event-based controllers [8] have also been proposed with the idea of computing new control actions only when necessary, i.e., triggered by specific events, leading to lower utilization of the computing and communication resources.

The implementation of controllers with flexible timing constraints is not straightforward in container-based virtualization that uses static resource reservation. The change in timing constraints must be reflected in the resource reservation for the real-time container hosting the virtualized controller. Additionally, container-based virtualization is known for its imperfect temporal isolation, which leads to timing disturbances in virtualized applications. These challenges in the current container-based virtualization technologies hinder a full-fledged utilization of them in time-sensitive control systems where the system dynamics have to be considered for a better Quality of Control (QoC).

This paper tackles the challenges mentioned above by proposing a resource adaptation mechanism for real-time containers to improve the QoC [9] for virtualized controllers while minimizing the resource utilization for computing new control actions. The main goal of the proposed mechanism is to dynamically adjust the timing constraints of a flexible controller together with the adaptation of the resource reservation for real-time containers.

The resource reservation has to reflect the change in the timing constraints in the sense that the shorter control period requires a higher resource reservation and vice versa for the longer control period. Adjustment of the timing constraints and resource reservation is done jointly to obtain improved resource utilization and, at the same time, to improve the QoC. In this setup, each control function resides in a real-time container. The proposed mechanism is based on existing work on flexible control, e.g., [6], [7], to be used together with dynamic resource reservation for real-time containers. The concrete contributions of this paper are as follows:

- We propose a mechanism to dynamically adapt the Control Application's timing constraints, including the control sampling rates, together with the resource reservation of real-time containers. Adjustment of timing constraints may lead to improved QoC [6].
- We implement the proposed mechanism in the hierarchical control group scheduler patch (HCBS) [10] on Linux. This enables Docker containers to host controllers with flexible timing constraints.
- We experimentally evaluate the performance of the proposed mechanism on an example of a Control Application. We demonstrate that such an adaptation, together with the resource reservation in real-time containers, will lead to

better responses to disturbances in the Control Application, as well as minimize response deviations.

This paper is structured as follows. Section II reviews the related literature. Section III presents the technical background on resource adaptation for real-time containers and on control using flexible constraints. Section IV proposed an adaptive virtualized controller, while Section V evaluates the feasibility of the adaptive virtualized controller. Finally, in Section VII we conclude the paper and present an outlook on future work.

II. Related Work

There have been several proposals related to support for real-time applications in container-based virtualization that can be a foundation for the virtualization of control applications. Moreover, several works have addressed the challenges of dynamic resource reservation for container-based virtualization. In addition, the field of control theory focuses on flexible timing constraints. In this section, we briefly present these proposals to position the contribution of this paper.

Resource adaptation for containers: Several works focus on resource adaptation (also known as vertical scaling) for containers in the presence of performance losses. For instance, the work in [11] proposes a framework that, through a controller hierarchy, dynamically adjusts the resources of real-time containers to match the required performance levels. In this work, the HCBS patch [10] is utilized to implement the adaptation mechanisms. Moreover, the work in [12] proposes ElasticDocker, a system that automatically scales Docker containers vertically based on the current workload of containerized applications. ElasticDocker monitors the performance of containers at runtime and dynamically changes resource reservations to adapt to the dynamic changes via a threshold-based algorithm. The work presented in [13] proposes metrics to measure the performance of real-time containers. The paper describes a container orchestration framework based on Kubernetes that uses real-time metrics for container placement decisions. Rossi et al. [14] uses reinforcement learning to enable adaptive runtime deployment without manually tuning container-based applications using horizontal and vertical elasticity. The work of Shekhar et al. [15] proposes a data-driven approach by machine learning to create predictive system performance runtime models that can be adapted to workload changes.

Controllers with flexible timing constraints: The work in [6] argues that static timing constraints hinder controllers from adapting to changing system dynamics, resulting in sub-optimal results since higher execution rates that can more quickly react to disturbances waste resources when the system is at an equilibrium. Furthermore, the work in [16] argues that there is a need to add flexibility in real-time control loops, since, previously, systems have been based on static analysis and design under the assumption that the controllers execute in a predictable (hardware and software) environment. However, the authors in [16] argue that hardware platforms and operating systems tend to be commercial-off-the-shelf (COTS) products with poor real-time specifications that are typically tailored to improve average-case performance rather than predictable worst-case performance.

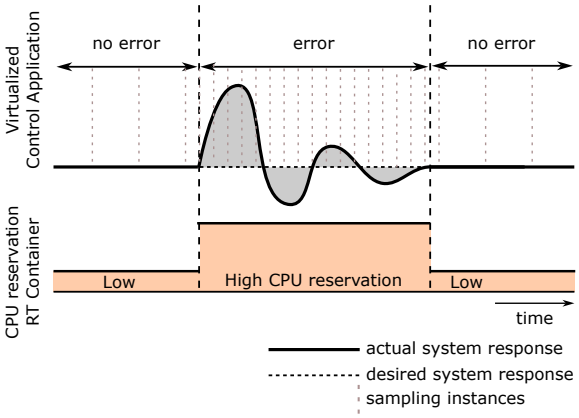

Fig. 1. The control period and resource reservation change in relation to the entity of the error (adapted from [6]).

Our work aims to bridge the gap between these two distinguished research areas of dynamic resource adaptation for container-based virtualization and control with flexible timing constraints to better utilize computational resources.

III. Technical Background

This section aims to provide a comprehensive overview of technical concepts related to the control with flexible timing constraints and resource reservation for real-time containers.

A. Control with flexible timing constraints

Classical closed-loop control applications that have a static execution rate assume that the sampling and actuation rates are constant over the lifetime of the system. The chain of (i) measuring the controlled variable, (ii) computing the control signal, and (iii) applying the computed value to the actuators is triggered by a single periodic source of events [17]. Not only is the choice of the rate of sensing, computing, and actuating crucial for the correct operation of the system but this rate must also be maintained between successive execution instances of the control loop in order to ensure system stability and meet the QoC objectives [5]. In principle, it may be possible to use different rates of execution for the control, but typically control system designers adhere to one value that has been computed at design time. Translated into real-time requirements, the sensing, control, and actuation actions are expressed as periodic tasks (with the task period being equal to the period of the control) with implicit or constrained deadlines that are mainly used to restrict the completion times of tasks [6]. Marti et al. [6] demonstrates that being able to adapt the rate of execution and, therefore, the timing constraints for control tasks can lead to better control by allowing faster reactions to transient disturbances. Fig. 1 illustrates control period instances and the error (represented by the shaded area) in the closed-loop system that was induced

by a perturbation. The figure shows that the controller executes a control algorithm more frequently at instants with a high quantity of perturbation.

B. Resource reservation for real-time containers

Techniques such as container-based virtualization have emerged as interesting mechanisms to be used in complex embedded systems [10], [18], [19]. Container-based virtualization uses mechanisms of the host operating system (e.g., cgroups in Linux) to provide spatial isolation of tasks and control their bandwidth access to, e.g., CPU and memory. Improving the timing predictability of container-based virtualization has been attempted, e.g., via PREEMPT_RT patch [20], [21], real-time co-kernel such as Xenomai [22], or RTAI [23]. Several real-time scheduling mechanisms can be employed to ensure temporal isolation amongst virtualized components sharing the same physical resource. One such approach is the Compositional Scheduling Framework [24], which is implemented in Linux as the Hierarchical Control Group Scheduler (HCBS) [10].

HCBS uses a reservation-based algorithm in which at each replenishment period (P_k), each container (k) is assigned a maximum budget (or runtime) Q_k. The scheduler guarantees that the container will be able to execute the reserved budget within the period but not exceed it, thus not interfering with the resource allocation of other co-located containers. HCBS uses static resource reservation that a system designer typically determines before the system starts. Therefore, the system cannot respond to dynamic changes such as changes in workload or the need to execute the controller at a higher (or lower) rate. Additionally, container-based virtualization is prone to timing disturbances due to shared platforms of co-located containers [25], [15], [26], [27]. In instances of changing workloads or interference from other components, static reservation may lead to timing violations. Conversely, not being able to adapt the rate of execution may lead to either insufficient resource usage or a degrading of QoC.

IV. QoC Controller

In this section, we present the design of a QoC controller targeting time-critical Control Applications. We design the QoC controller to automatically change the timing constraint of the virtualized application and, at the same time, adjust the container's resource reservation to the varying QoC metrics expressing the performance of the virtualized Control Application. The overall concept is presented in Fig. 2, which consists of four elements. These elements include (i) a real-time (RT) container that encompasses the Control Application, (ii) the Control Application itself that is responsible for controlling the system under control, (iii) the Plant, which is the target system to be controlled by the Control Application, and (iv) the QoC Controller which is responsible for adaptation in the resources and timing constraints for improving the performance of the Control Applications. Following, we describe the elements in more detail.

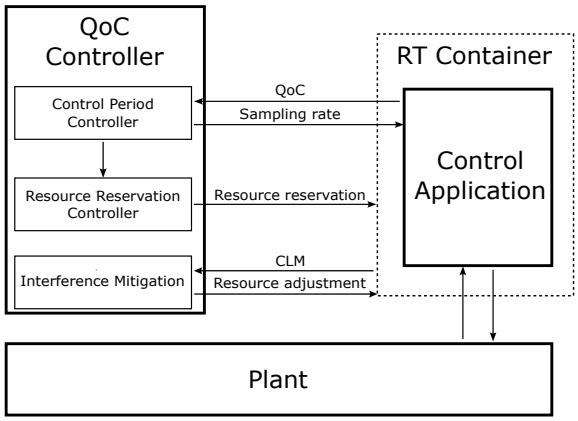

Fig. 2. System overview containing QoC Controller and its modules, real-time container hosting a Control Application, and a plant.

Real-time container (RT container): Real-time containers provide a separate and secure environment for running software applications, and they are designed to ensure the temporal predictability of the system. We use the HCBS patch [10] for the implementation of real-time containers that allow for a static reservation of computing resources. We use Docker, which is a platform that enables the packaging, configuration, and execution of applications via container-based virtualization technology (e.g., utilizing Linux features cgroups and namespaces). Developers can bundle applications and their dependencies into containers that can be ported and run on any machine that has Docker installed. This makes it easy to deploy in various environments. In terms of real-time behavior, each of the real-time containers (k) has a CPU reservation Q_k that is replenished every period P_k, as described above. Once the budget is exhausted, the container will be suspended until the next replenishment period.

In order to ensure the schedulability of the system, the following condition must be fulfilled $\sum_{k=1}^{n} P_k/Q_k \leq 1$, where n is a number of co-located containers in a physical platform. In our case, schedulability refers to all containers being able to execute until their deadlines, which are assumed to be implicit, i.e., they are equal to the container period.

The Control Application: The Control Application is implemented as a periodic real-time task that performs computations at predefined intervals (not only a single interval) determined by the system designer [17]. The shorter the interval, the better the QoC, and the more computational power required. This can be a limiting factor for embedded systems with limited resources. Nevertheless, this is a typical trade-off in real-time systems in which more resources will lead to better performance; however, the available resources are typically limited. Therefore, resource management becomes an important issue to handle, in particular dynamically, to support the system's dynamicity.

The Plant: The term *plant* refers to the physical instance or mathematical model of the system being controlled. The

continuous-time behavior of the plant is sampled at regular intervals through sensors.

QoC Controller The primary objective of the QoC controller, as depicted in Fig. 2, is to optimize the performance of the Control Application through the execution of four distinct functions: (i) a function to adjust timing constraints of the virtualized Control Application based on its QoC metrics, (ii) a container's resource reservation, (iii) interference mitigation, and (iv) communication with co-located QoC controllers.

Timing constraints adjustment refers to the adaptation of control periods for control applications based on QoC metrics. It refers to the adjustment of the control period. In particular, QoC can be improved by sampling the state of the plant more frequently by decreasing the period of control, resulting in more precise control actions. This is particularly useful for handling transients. However, if the QoC is fairly good, the frequency at which the controller takes action may be decreased, which in turn releases computing resources for other co-located controllers.

The resource allocation function is responsible for adjusting the time constraints of the container resource reservation based on current requirements and real-time performance. It can either increase or decrease the resource allocation accordingly. This function dynamically adjusts the allocation of CPU resources to meet the changing demand of the Control Application. The interference mitigation function aims to compensate for performance losses caused by co-located containers, which can result in unpredictable interference, and to optimize the provisioned CPU resources. In the next section, we provide a detailed analysis of the performance of these functions within the QoC controller. Lastly, the communication function ensures fair distribution of CPU resources among co-located QoC controllers and prevents the over-reservation of available resources.

A. Control Period Controller

The Control Period Controller continuously observes the Control Application's QoC metric and, based on this metric, the controller determines the control period.

QoC metric measurement: The performance of the Control Application can be expressed as a QoC cost function computed as follows:

$$\text{QoC}(t) = \int_{t-h}^{t} \|y_{\text{des}}(t) - y_{\text{act}}(t)\|^2 \qquad (1)$$

where, $y_{\text{des}}(t)$ is the setpoint, i.e., the desired behavior of the controlled variable $y_{\text{act}}(t)$, and $t - h$ is the time the last control action has been taken. Note that high values of QoC correspond to poor controller performance and low values of QoC correspond to good performance.

The controller determines the difference between the current (y_{act}) and desired (y_{dest}) states of the controller system, then generates a control signal with the aim of bringing the difference to zero. The instantaneous value of QoC can be used to evaluate the current performance of the controller, and it can be used to adjust the control period of the controller.

TABLE I
THRESHOLD OF QOC CONTROLLER.

QoC	Control Period	Container Budget
Low	Very High	Very High
Normal	Normal	Normal
High	Low	Low
Very High	Very Low	Very Low

Control period calculation: To implement the controller, we chose a threshold controller as our control system. A threshold controller functions by comparing a measured quantity, specifically the QoC, to a predefined threshold or setpoint value. When the measured quantity exceeds the threshold value, the controller initiates appropriate actions to modify the system and bring the measured quantity back within the desired range. The thresholds for the QoC controller are described in Table I. We assume that the thresholds are defined by a system designer.

B. Resource Reservation Controller

Based on the decision of the Control Period Controller, the resource reservation controller adjusts the resource reservations for the real-time container. It adjusts the resource reservation by modifying the period and budget of the containers.

Similarly to the Control Period Controller, the Resource reservation controller employs threshold-based rules to provision and de-provision resources for the RT containers, as shown in Table I.

C. Interference Mitigation

The resource adjustment function acts as a bridge between the ideal container resource reservation, which is based on simplified models, and the actual dynamic behavior of the containers. The performance of containers can be negatively impacted by the presence of co-located containers. This issue becomes particularly significant in systems like the one examined in this article, where co-located containers experience significant variations in their workload. When containers are colocated on shared platforms, they may compete for shared resources that may introduce timing disturbances, such as cache misses, memory and bus contention, and translation look-aside buffers [15], which are discussed in [11].

Hence, the resource adaptation function in the system dynamically modifies the CPU resources allocated to real-time containers by continuously assessing their real-time performance. Its primary objective is to counteract the unforeseen interference caused by co-located containers. To evaluate real-time performance, the system uses container-level metrics (CLM) introduced in a previous work [13]. These metrics include response time and missed deadlines. By measuring and readjusting the CPU bandwidth reservation for containers, this function compensates for performance losses within a computing node. Its purpose is to ensure that the control application meets the specified response time in the presence of interference that may cause the execution time to exceed

the worst-case assumption that was used to check the system schedulability.

D. Communication with other controllers

Due to the limited CPU resources on physical platforms, it is essential for the system to monitor the available CPU resources that can be allocated among real-time (RT) containers based on predefined policies. The problem is the fact that there are limited resources if multiple co-located Control Applications face an increase in QoC values, and the controller can not independently increase the control period. As depicted in Fig. 3, there are multiple QoC controllers that adjust the control period and CPU resources for its corresponding real-time containers.

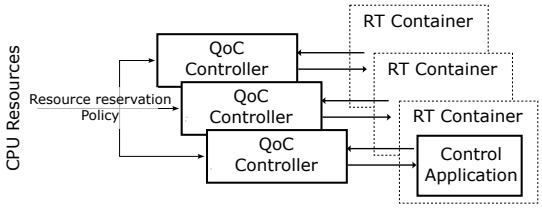

Fig. 3. Multiple controllers.

V. Experimental Evaluation

In this section, we present an experiment that evaluates the feasibility of controlling the control periods and CPU resources based on the QoC values. We conducted an experimental demonstration of the adaptation process utilizing an Intel i5 computer equipped with 8GB of RAM, running Debian Linux (Kernel version 5.2.8) that was patched with the HCBS, and running Docker v20.10.

In this experiment, we used the inverted pendulum use case to show the performance of the designed control system. An inverted pendulum is a widely studied use-case in control theory, which involves a pendulum anchored on a platform (cart) that can move along a track in one dimension (along the x-axis). The governing equations for the inverted pendulum given in the literature (e.g. [28, p. 43]) are:

$$\begin{cases} (M+m)\ddot{x} + b\dot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta = F \\ (I+ml^2)\ddot{\theta} + mgl\sin\theta = -ml\ddot{x}\cos\theta \end{cases} \quad (2)$$

where M are the masses of the cart and pendulum, respectively, b represents the friction coefficient, l is the length of the pendulum, I represents the moment of inertia of the pendulum, F is the force that is applied to the entire cart, x is the position of the cart, and θ is the angle of the pendulum beam. The pendulum is balanced in an inverted position, which means that its center of mass is directly above the pivot point. The control loop tries to maintain the inverted pendulum in balance while also maintaining a given position of the cart by actuating the force applied to the cart along the x-axis. We use a virtualized PID controller as a Control Application. Fig. 5 illustrates a system consisting of an uncontrolled inverted pendulum.

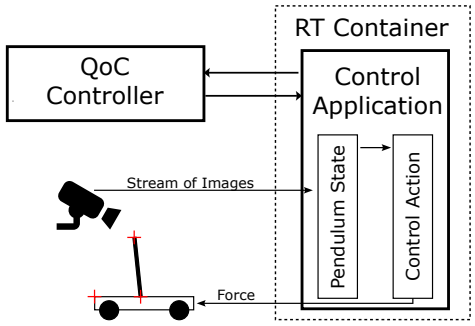

Fig. 4. Inverted pendulum experiment.

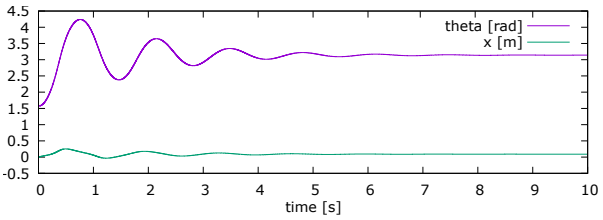

Fig. 5. Inverted pendulum use-case.

The setting of the inverted pendulum experiment is depicted in Fig. 4. We developed a pendulum simulator that is equipped with a (virtual) camera that produces images of the pendulum. The camera streams the pendulum images into the controlled system. The controller utilizes image processing algorithms to analyze the image stream and infer the state of the pendulum. The stream of images is processed by the OpenCV library, which is a computer vision toolkit that offers a range of functions and algorithms specifically designed for real-time object detection and tracking. Using the OpenCV library, the controller captures the image feed from a camera and applies image-processing techniques to detect the position and angle of the pendulum. This information is used to calculate the appropriate control actions to keep the pendulum upright. The algorithm uses a template matching to locate the tip of the pendulum, its base, and the left corner of the cart as indicated in Fig. 4. The processing of the images is a non-trivial action that requires properly allocated CPU resources.

The results of the inverted pendulum experiment are illustrated in Fig. 6. In all sub-figures, the y-axis presents the measured values, while the x-axis is the time instance we measured each value. In this experiment, we have the pendulum not in the upright position at the beginning. Then, after equilibrium, at time instance 300, we intentionally make the pendulum unbalanced again to see how the QoC value develops again.

Based on this experiment, Fig. 6.a shows the QoC values and instant errors of the control. Initially, the pendulum is not situated in the desired upright and balanced position, leading to a significant error between the desired and actual state. As time progresses, the pendulum converges towards its setpoint, resulting in the reduction of the error and QoC. At

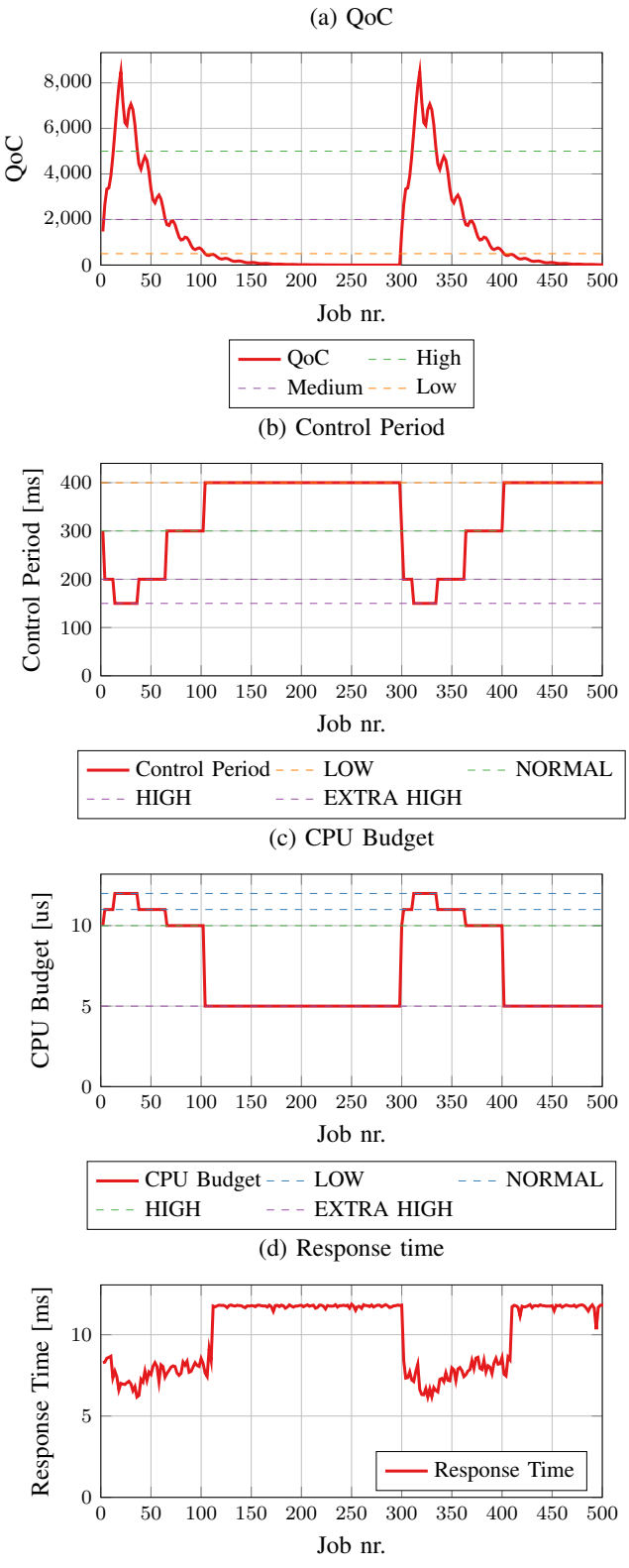

Fig. 6. Experimental results for the inverted pendulum experiment.

time instance 300 the pendulum is perturbed, leading to an increase in error and QoC again. We divided the QoC range into four thresholds that classify the level of QoC: Low (for QoC lower than 500), Normal (when it is 2000), High (when it is 5000), and Very High (QoC higher than 5000). Note that these values are selected based on the use-case which can vary depending on the system under control. The development of QoC values shows that the quality can reach the Low range when the pendulum becomes stable. In general, Fig. 6.a shows that the Control Application within the container performs as expected.

Fig. 6.b shows the corresponding control period. The value for the control period is based on the QoC value. With the decreasing QoC value, while the pendulum is converging to equilibrium, the control period increases. This will reduce computing resource utilization as there is no need for frequent control of the system. At time instance 300, the control period decreases again to perform a better control on the pendulum, which is not in equilibrium again. This is necessary because there is a need for more frequent control of the system.

Fig. 6.c shows the CPU budget. Similarly to the control period, the CPU budget value is based on the QoC value. With decreasing QoC value, the CPU budget reservation decreases. The CPU budget decreases to decrease the computing resource utilization when the pendulum becomes stable, i.e., there is no need for frequent control. Note that increasing the period and decreasing the budget significantly reduces computing resource utilization. Finally, Fig. 6.d shows the response time of the container that consists of the image processing part and the control action. The value depends on the CPU budget reservation. Allocating more CPU budget decreases the response time.

VI. Implementation on Linux

In order to demonstrate the feasibility and behavior of the system, we have integrated the suggested QoC-aware controller into the Debian GNU/Linux 10 (buster) operating system patched with the HCBS patch. This patch enables the management of resources for containers during runtime without the need for modifications of the container runtime. We have enhanced the Linux Kernel by incorporating the adaptability of CPU resources for RT containers based on the QoC values. While the HCBS patch allows static configuration of CPU budget and CPU period for containers, our supplementary modifications enable dynamic adjustment of CPU resources for RT containers during runtime. The cornerstone of the Kernel extension is a set of custom syscalls that are being called from a containerized Control Application. Based on the syscalls, the Kernel adjusts the control period and resource reservation. The containerized application is a C application that generates an infinite sequence of jobs. Each job measures controlled variables and computes the value of the control signal. After the completion of the job, the application is suspended until the time of the next job activation (using *clock_nanosleep()*). The application communicates with the Linux kernel via syscalls, which computes error and QoC

then actuates the CPU budget for the real-time containers and control period. An overview of custom syscalls is as follows:

- **Application Initialization**: It is called after the application is started and sets initializes variables and constants for the control mechanism in Linux Kernel: QoC values, the QoC thresholds, periods, and CPU budgets.
- **Job Activation**: This syscall is called right after a job is activated (waked up). It captures the job activation time, that is used for computing job response time.
- **Job Finished**: It is called on the completion of the job. It computes error as the difference between the desired and actual states. And computes current QoC as shown in Equation 1.

VII. Conclusions and future work

In this paper, we presented a resource adaptation approach that takes into account the QoC for real-time containers. Container-based virtualization offers significant benefits for the deployment of industrial applications. However, addressing challenges such as ensuring complete temporal isolation among real-time applications and supporting time-critical applications becomes crucial, particularly in scenarios with varying workloads. This paper proposes a mechanism to dynamically adapt timing constraints and resource reservations in order to improve the QoC for time-sensitive control applications. The proposed mechanism is implemented and evaluated on a Linux-based hierarchical scheduling platform. The results obtained from the evaluation demonstrate the feasibility of our approach, where the controller modifies the timing constraints of the virtualized control application and updates the resource reservation for the hosting container accordingly. With an experiment using an inverted pendulum, we show that it is possible to control the pendulum while simultaneously adjusting the utilization of computing resources during the control process. This capability has a direct impact on optimizing computing resource usage.

There are several interesting directions for future work. In this paper, we have presented a resource adaptation mechanism for real-time containers considering the QoC. However, more complex control, decision, and predictive algorithms are needed to capture more complex scenarios. An ongoing work targets systems that have multiple virtualized controllers.

References

[1] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *Euromicro Int. Conf. on Par., Distr., and Netw. Proc. (PDP)*, 2013.

[2] M. G. Xavier, M. V. Neves, and C. A. F. De Rose, "A performance comparison of container-based virtualization systems for mapreduce clusters," in *Euromicro Int. Conf. on Par., Distr., and Netw. Proc. (PDP)*, 2014.

[3] F. Ramalho and A. Neto, "Virtualization at the network edge: A performance comparison," in *IEEE Int. Symp. A World of Wirel., Mob. and Multim. Net. (WoWMoM)*, 2016.

[4] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," in *IEEE Int. Symp. on Perf. Analysis of Syst. and Soft. (ISPASS)*, 2015.

[5] B. Wittenmark, K. J. Åström, and K.-E. Årzén, "Computer control: An overview," *IFAC Professional Brief*, 2002.

[6] P. Marti, J. M. Fuertes, G. Fohler, and K. Ramamritham, "Improving quality-of-control using flexible timing constraints: metric and scheduling," in *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.* IEEE, 2002.

[7] G. Fohler, "Dynamic timing constraints - relaxing overconstraining specifications of real-time systems," 2001.

[8] K. J. Åström, *Event Based Control*. Springer Berlin Heidelberg, 2008.

[9] M. Barzegaran, A. Cervin, and P. Pop, "Towards quality-of-control-aware scheduling of industrial applications on fog computing platforms," in *Proceedings of the Workshop on Fog Computing and the IoT*, ser. IoT-Fog '19. Association for Computing Machinery, 2019.

[10] L. Abeni, A. Balsini, and T. Cucinotta, "Container-based real-time scheduling in the linux kernel," *ACM SIGBED Review*, 2019.

[11] V. Struhár, S. S. Craciunas, M. Ashjaei, M. Behnam, and A. V. Papadopoulos, "Hierarchical resource orchestration framework for real-time containers," 2023.

[12] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Autonomic vertical elasticity of docker containers with elasticdocker," in *2017 IEEE 10th international conference on cloud computing (CLOUD)*. IEEE, 2017.

[13] V. Struhár, S. S. Craciunas, M. Ashjaei, M. Behnam, and A. V. Papadopoulos, "React: Enabling real-time container orchestration," in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2021.

[14] F. Rossi, M. Nardelli, and V. Cardellini, "Horizontal and vertical scaling of container-based applications using reinforcement learning," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019.

[15] S. Shekhar, H. Abdel-Aziz, A. Bhattacharjee, A. Gokhale, and X. Koutsoukos, "Performance interference-aware vertical elasticity for cloud-hosted latency-sensitive applications," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018.

[16] A. Cervin, "Integrated control and real-time scheduling," Ph.D. dissertation, Univ., 2003.

[17] A. Leva and A. V. Papadopoulos, "Tuning of event-based industrial controllers with simple stability guarantees," *Journal of Process Control*, 2013.

[18] A. Celesti, D. Mulfari, M. Fazio, M. Villari, and A. Puliafito, "Exploring container virtualization in iot clouds," in *2016 IEEE international conference on Smart Computing (SMARTCOMP)*. IEEE, 2016.

[19] A. V. Papadopoulos, M. Maggio, A. Leva, and E. Bini, "Hard real-time guarantees in feedback-based resource reservations," *Real-Time Systems*, 2015.

[20] A. Moga, T. Sivanthi, and C. Franke, "Os-level virtualization for industrial automation systems: are we there yet?" in *SAC '16*, 2016.

[21] T. Goldschmidt, S. Hauck-Stattelmann, S. Malakuti, and S. Grüner, "Container-based architecture for flexible industrial control applications," 2018.

[22] T. Tasci, J. Melcher, and A. Verl, "A container-based architecture for real-time control applications," in *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. IEEE, 2018.

[23] M. Cinque, R. Della Corte, A. Eliso, and A. Pecchia, "Rt-cases: Container-based virtualization for temporally separated mixed-criticality task sets," in *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, 2019.

[24] I. Shin and I. Lee, "Compositional real-time scheduling framework," in *25th IEEE International Real-Time Systems Symposium*. IEEE, 2004.

[25] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, 2015.

[26] S. K. Garg and J. Lakshmi, "Workload performance and interference on containers," in *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation*, 2017.

[27] P. Sharma, L. Chaufournier, P. Shenoy, and Y. Tay, "Containers and virtual machines at scale: A comparative study," in *Proceedings of the 17th international middleware conference*, 2016.

[28] G. Conte, C. Moog, and A. Perdon, *Algebraic Methods for Nonlinear Control Systems*, ser. Communications and Control Engineering. Springer London.