Proceedings of

RAGE 2022

The 1st *Real-time And intelliGent Edge computing workshop*

July 10th, 2022, San Francisco, CA, USA

In conjunction with

The 59th Design and Automation Conference
July 10-14, San Francisco, CA, USA

Proceedings Edited by:

Daniel Casini
Dakshina Dasari
Matthias Becker

the organizers and program chairs of RAGE 2022.

Message from the Chairs

Welcome to the first edition of the Real-Time And intelliGent Edge computing workshop (RAGE 2022), which is held in conjunction with the 59th edition of the Design and Automation Conference (DAC 2022). The workshop takes place in San Francisco, California, USA, on July 10th, 2022. The workshop is composed of *nine* high-end invited speakers and *eight* papers. Invited speakers are Anthony Rowe from the Carnegie Mellon University, Arne Hamann from Bosch, Joerg Seitter from ETAS, Giorgiomaria Cicero from Accelerat Srl, Frédéric Desbiens from the Eclipse Foundation, Pratham Oza from NuroAI, Arpan Gujarati from the University of British Columbia, Hana Khamfroush from the University of Kentucky, and Mohammad Al Faruque from the University of California at Irvine. Among the eight papers, two of them are invited contributions. The first one is from Matteo Maria Andreozzi and Girish Shirasat from ARM, and the second one is from Hyunjong Choi, Daniel Enright, Hoora Sobhani, Yecheng Xiang, and Hyoseung Kim from the University of California at Riverside. All contributions discuss different aspects of edge computing, such as resource allocation, communication protocols, industrial use cases for edge computing, and others.

Two contributions from the open call for papers have been awarded the *best paper* and *best presentation* awards. The best paper award has been given to the paper titled *Minimal-Overlap Centrality Driven Designation for Real-Time TSCH Networks*, by Miguel Gutiérrez Gaitán, Pedro d'Orey, Pedro Santos, and Luís Almeida. The best presentation award has been given to Andres Meza for the presentation of the paper titled *Safety Verification of Third-Party Hardware Modules via Information Flow Tracking*, authored by Andres Meza, Francesco Restuccia, Ryan Kastner, and Jason Oberg.

RAGE 2022 would not have been possible without the support of many people. We thank the DAC 2022 Executive Committee and Chia-Lin Yang, the 59th DAC Workshop Chair, which agreed to host this first edition in conjunction with DAC. We particularly thank Chia-Lin Yang and Alexis Bauer Kolak for their support in the organization of the workshop. We thank the authors and invited speakers for providing exciting talks and the RAGE 2022 program committee for reviewing papers and providing helpful feedback to authors. We finally thank the audience of the workshop for their interest and for the questions and discussion.

As the first edition of a new workshop, we hope this will give rise to a successful series of editions on an emerging and stimulating research topic.

The Workshop Chairs,

Daniel Casini
Scuola Superiore Sant'Anna,
Pisa, Italy

Dakshina Dasari
Robert Bosch GmbH,
Renningen, Germany

Matthias Becker
KTH Royal Institute of Technology,
Stockholm, Sweden

Program Committee

Takuya Azumi, *Saitama University, Japan*
Alessio Balsini, *Google, UK*
Soroush Bateni, *University of Texas at Dallas, USA*
Tobias Blass, *Apex.AI*
Giorgio Buttazzo, *Scuola Superiore Sant'Anna, Italy*
Albert Cheng, *University of Houston, USA*
Hyunjong Choi, *University of California at Riverside, USA*
Xiaotian Dai, *University of York, UK*
Zheng Dong, *Wayne State University, USA*
Pierfrancesco Foglia, *University of Pisa, Italy*
Gabriel Parmer, *George Washington University, USA*
Miguel Gutiérrez Gaitán, *CISTER, Portugal*
Naresh Nayak, *Robert Bosch GmbH, Germany*
Alessandro Vittorio Papadopoulos, *Mälardalen University, Sweden*
Paolo Pazzaglia, *Universität des Saarlandes, Germany*
Carlo Puliafito, *University of Pisa, Italy*
Francesco Restuccia, *UC San Diego, CA, USA*
Juan M. Rivas, *Universidad de Cantabria, Spain*
Claudio Scordino, *Evidence Srl, Italy*
Biruk B. Seyoum, *Columbia University, USA*

Publicity Chair

Francesco Restuccia, *UC San Diego, CA, USA*

Web Chair

Gabriele Serra, *Scuola Superiore Sant'Anna, Pisa, Italy*

AGENDA

WORKSHOP PROGRAM

Sunday, 10th July 2022 - Morning session	
08:00-08:10	Welcome message from the chairs
08:10-08:40	**Invited Talk:** Lightweight virtualization for giving the cloud an edge **Prof. Anthony Rowe,** Carnegie Mellon University, USA
08:40-09:25	Session 1: Tools, architectures, and resource allocation for the edge.
08:40-08:55	MArK8s - A Management Toolchain Approach for Automotive Real-Time Kubernetes Containers in the Mobile Edge Cloud [ ] *Bernhard Blieninger, Aaron Dietz and Uwe Baumgarten*
08:55-09:10	Towards a Predictable and Cognitive Edge-Cloud Architecture for Industrial Systems [ ] *Mohammad Ashjaei, Saad Mubeen, Masoud Daneshtaalb, Victor Casamayor and Geoffrey Nelissen*
09:10-09:25	RT-SCALER: Adaptive Resource Allocation Framework for Real-Time Containers [ ] *Vaclav Struhar, Silviu S. Craciunas, Mohammad Ashjaei, Moris Behnam and Alessandro Papadopoulos*
09:25-09:40	Short Break
09:40-10:10	**Invited Talk:** Industrial use-cases for real-time edge-computing **Dr. Arne Hamann,** Bosch Corporate Research, Germany
10:10-10:40	Session 2: **Invited speakers with papers.**
10:10-10:25	High-performance real-time systems design from cloud to embedded edge. [ ] *Matteo Andreozzi and Girish Shirasat, ARM.* *PRESENTED BY* ***Dr. Girish Shirasat,*** *ARM, UK*
10:25-10:40	Priority-Driven Real-Time Scheduling in ROS2: Potential and Challenges [ ] *Hyunjong Choi, Daniel Enright, Hoora Sobhani, Yecheng Xiang and Hyoseung Kim.* *PRESENTED BY* ***Dr. Hyunjong Choi,*** *University of California at Riverside, USA*
10:40-10:55	Short Break
10:55-11:25	**Invited Talk:** Giving the Software Defined Vehicle an Edge

	Joerg Seitter, ETAS, Germany
11:25- 11:55	**Invited Talk:** The role of virtualization at the edge for mixed-criticality applications **Giorgiomaria Cicero,** Accelerat S.R.L., Italy
11:55- 12:55	Lunch

Sunday, 10th July 2022 - Afternoon session	
12:55- 13:25	**Invited Talk:** zenoh: A Next-Generation Protocol for IoT and Edge Computing **Dr. Frédéric Desbiens,** Eclipse Foundation, Canada
13:25- 14:10	Session 3: Real-time networks, safety, and queueing delays.
13:25- 13:40	Minimal-Overlap Centrality-Driven Gateway Designation for Real-Time TSCH Networks [ ] *Miguel Gutiérrez Gaitán, Pedro d'Orey, Pedro Santos and Luís Almeida*
13:40- 13:55	No-more-unbounded-blocking queues: bounding transmission latencies in real-time edge computing [ ] *Gabriele Serra and Pietro Fara*
13:55- 14:10	Safety Verification of Third-Party Hardware Modules via Information Flow Tracking [ ] *Andres Meza, Francesco Restuccia, Ryan Kastner and Jason Oberg*
14:10- 14:25	Short Break
14:25- 14:55	**Invited Talk:** Deadline-Aware Task Offloading for Vehicular Edge Computing Networks **Dr. Pratham Oza,** Nuro Inc., USA
14:55- 15:25	**Invited Talk:** Serving DNNs like Clockwork: Performance Predictability from the Bottom Up **Dr. Arpan Gujarati,** University of British Columbia, Canada
15:25- 15:40	Short Break
15:40- 16:10	**Invited Talk:** QoS-aware resource management for Edge-AI **Prof. Hana Khamfroush,** University of Kentucky, USA
16:10- 16:40	**Invited Talk:** Low power Machine Learning Techniques for Edge-AI

	Prof. Mohammad Al Faruque, University of California at Irvine (UCI), USA
16:40-17:10	Panel Discussion
17:10-17:30	Closing Remarks

CONTENTS

Invited Talk

Lightweight virtualization for giving the cloud an edge

Prof. Anthony Rowe,
Carnegie Mellon University, USA

Prof. Anthony Rowe is the Siewiorek and Walker Family Professor in the Electrical and Computer Engineering Department at Carnegie Mellon University. His research interests are in networked real-time embedded systems with a focus on wireless communication. He has worked on topics including large-scale sensing for critical infrastructure monitoring, indoor localization, building energy-efficiency and technologies for microgrids. His most recent work has looked at connecting embedded sensing systems with mixed reality and spatial computing platforms. He is currently the director of the SRC/DARPA sponsored CONIX Research Center which spans seven Universities with the goal of exploring future distributed computing architectures. His past work has led to dozens of hardware and software systems, seven best paper awards, talks at venues like the World Economic Forum in Davos and several widely adopted open-source research platforms. He earned a Ph.D in Electrical and Computer Engineering from CMU in 2010, received the Lutron Joel and Ruth Spira Excellence in Teaching Award in 2013, the CMU CIT Early Career Fellowship and the Steven Fenves Award for Systems Research in 2015 and the Dr. William D. and Nancy W. Strecker Early Career chair in 2016.

Invited Talk

Industrial use-cases for real-time edge-computing

Dr. Arne Hamann,
Bosch Corporate Research, Germany

Dr. Arne Hamann (Bosch) obtained his PhD in Computer Science in 2008 from the Technical University of Braunschweig Germany. He is Chief Expert for "Distributed Intelligent Systems" at Bosch Corporate Research. Like the Bosch product portfolios his range of actives is very broads encompassing complex embedded systems where the interaction between physical processes hardware and software plays a major role through to distributed IoT systems with elements of (edge) cloud computing. In the academic contexts he is member of the editorial board of the ACM journal “Transactions on Cyber Physical Systems” and regularly serves as program committee member for international conferences such as ECRTS, RTSS, RTAS, DAC, EMSOFT, and ICCPS.

Invited Talk

Giving the Software Defined Vehicle an Edge

Joerg Seitter,
ETAS, Germany

Joerg is heading the advance engineering of ETAS with a focus on Reliable Distributed Systems. In former roles within the Bosch group, he was working in Automotive Software and System Architecture and lead the development of high-performance ECUs for powertrain systems. Before Bosch he was working at IBM and has a deep history in large scale database systems technology and holds a lecture on this topic at University of applied science Esslingen. He holds a B.Eng. from University of applied Science Esslingen and a M.Sc. Degree from Brunel University London.

Invited Talk

The role of virtualization at the edge for mixed-criticality applications

Giorgiomaria Cicero,
Accelerat S.R.L., Italy

Giorgiomaria Cicero is CEO at Accelerat Srl (https://accelerat.eu/) and Senior Research Fellow at the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna of Pisa. Accelerat is a spin-off company of Scuola Superiore Sant'Anna focused on software solutions for safe, secure, and time-predictable cyber-physical systems. The company was born as a technology transfer effort from the ReTiS Laboratory of Scuola Superiore Sant'Anna, one of the world's leading research teams in the area of embedded real-time systems. Giorgiomaria has a B.Sc. in Computer Engingeering, M.Sc. in Embedded Computing Systems, and has been visiting trainee at the European Space Agency (ESTEC, Netherlands). His research interests include software predictability in multi-processor systems and heterogeneous platforms, system-level cyber-security hardening techniques, and design and implementation of real-time operating systems and hypervisors.

Invited Talk

zenoh: A Next-Generation Protocol for IoT and Edge Computing

Frédéric Desbiens,
Eclipse Foundation, Canada

Frédéric Desbiens manages IoT and Edge Computing programs at the Eclipse Foundation. His job is to help the community innovate by bringing devices and software together. He is a strong supporter of open source. He worked as a product manager, solutions architect, and developer for companies as diverse as Pivotal, Cisco, and Oracle. Frédéric holds an MBA in electronic commerce, a BASc in Computer Science, and a BEd, all from Université Laval.

Invited Talk

Deadline-Aware Task Offloading for Vehicular Edge Computing Networks

Dr. Pratham Oza,
Nuro Inc., USA

Dr. Pratham Oza has a Ph.D. from Virginia Tech with research interests in cyber-physical systems and intelligent transportation with focuses on hardware/ software co-design of systems in the intersection of real-time systems and autonomous transportation. Pratham currently works as a Systems Engineer at Nuro with their autonomy division. Pratham has multiple academic and industry research collaborations and has won a Best Paper Award at IEEE RTCSA 2019. He is currently interested in analyzing the real-time requirements for vehicular applications relying on edge/ cloud connectivity. He is also an active contributor to the transportation research community serving as a reviewer and the president of the Institute of Transportation Engineers - Virginia Tech chapter.

Invited Talk

Serving DNNs like Clockwork: Performance Predictability from the Bottom Up

Dr. Arpan Gujarati,
University of British Columbia, Canada

Dr. Arpan Gujarati is Research Associate in the CS department at the University of British Columbia (UBC) in Vancouver (Canada). He is affiliated with the Systopia Lab at UBC, where he works with Margo Seltzer. Earlier, he spent a year as a postdoctoral researcher at the Max Planck Institute for Software Systems (MPI-SWS) in Saarbrücken (Germany), during which he worked with Jonathan Mace in the Cloud Software Systems Group. He completed his PhD thesis titled – Towards "Ultra-Reliable" CPS: Reliability Analysis of Distributed Real-Time Systems – under the supervision of Björn B. Brandenburg in the Real-Time Systems Group at MPI-SWS. He is the recipient of the 2021 SIGBED Paul Caspi Memorial Dissertation Award. He is broadly interested in real-time systems, distributed systems, fault tolerance, reliability analysis, and scheduling problems in the cloud domain as well as in the cyber-physical systems (CPS) domain.

Invited Talk

QoS-aware resource management for Edge-AI

Prof. Hana Khamfroush,
University of Kentucky, USA

Dr. Hana Khamfroush is an assistant professor at the computer science department of University of Kentucky, USA since 2018. Prior to this, she held a postdoctoral position at the computer science department of Penn State University, USA, between 2015 and 2017. Dr. Khamfroush's research interests include edge intelligence, wireless networks, network modeling and optimization. Her research has been funded by several sources and organizations including the National Science Foundation (NSF), Cisco Research Inc. and University of Kentucky. Dr. Khamfroush is a senior member of IEEE and a recipient of several awards, including two rising stars at EECS by MIT and CMU, and a Heidelberg Forum award. Beside her technical work, she has a passion for promoting underrepresented communities and specially women in computer science. She is the faculty advisor of ACM-W at the University of Kentucky and has received a prestigious Sarah Bennet Holmes award from the University of Kentucky for her contributions to issues that affects women in Kentucky.

Invited Talk

Low power Machine Learning Techniques for Edge-AI

Prof. Mohammad Al Faruque,
University of California at Irvine (UCI), USA

Prof. Mohammad Al Faruque is currently with the University of California Irvine (UCI), where he is an associate professor (with tenure) and directing the Cyber-Physical Systems Lab. Prof. Al Faruque is the recipient of the School of Engineering Mid-Career Faculty Award for Research 2019, the IEEE Technical Committee on Cyber-Physical Systems Early-Career Award 2018, and the IEEE CEDA Ernest S. Kuh Early Career Award 2016. He is also the recipient of the UCI Academic Senate Distinguished Early-Career Faculty Award for Research 2017 and the School of Engineering Early-Career Faculty Award for Research 2017. He served as an Emulex Career Development Chair from October 2012 till July 2015. Before, he was with Siemens Corporate Research and Technology in Princeton, NJ.His current research is focused on the system-level design of Internet-of-Things (IoT), Embedded Systems, and Cyber-Physical-Systems (CPS) with special interests on design automation methodologies, data-driven modeling techniques including machine learning for design, CPS security, etc. His work involves novel hardware and software design for various CPS application areas, including mobile health (mHealth), Industry 4.0 (manufacturing), smart-grid, and autonomous vehicles. Prof. Al Faruque received the Thomas Alva Edison Patent Award 2016 from the Edison Foundation, the 2016 DATE Best Paper Award, the 2015 DAC Best Paper Award, the 2009 IEEE/ACM William J. McCalla ICCAD Best Paper Award, the 2016 NDSS Distinguished Poster Award, the 2008 HiPEAC Paper Award, the 2015 Hellman Fellow Award, the 2015 Kane Kim Fellowship Award, the 2017 ICCAD Best Paper Award Nomination, the 2017 DAC Best Paper Award Nomination, the 2012 DATE Best IP Award Nomination, the 2005 DAC Best Paper Award Nomination, the EECS Professor of the year 2015-16 Award, and the 2015 UCI Chancellor's Award for Excellence in Fostering Undergraduate Research. Besides 100+ IEEE/ACM publications in the premier journals and conferences, Prof. Al Faruque holds 9 US patents. Prof. Al Faruque has published 2 books in the area of Embedded and Cyber-Physical Systems.

Mark8s - A Management Approach for Automotive Real-Time Kubernetes Containers in the Mobile Edge Cloud

Bernhard Blieninger
Automotive System Software and Architecture
fortiss GmbH
Munich, Germany
blieninger@fortiss.org

Aaron Dietz
Department of Informatics
Technical University Munich
Munich, Germany
aaron.dietz@tum.de

Prof. Dr. Uwe Baumgarten
Department of Informatics
Technical University Munich
City, Germany
baumgaru@tum.de

***Abstract*—This paper presents a management approach for real-time Kubernetes clusters in the automotive mobile edge cloud. As part of a vehicle-centric approach to future autonomous mobility the toolchain presented is positioned to extend, offload and enhance the computational capabilities for real-time tasks of a vehicle to the mobile edge cloud. The sensing of the environment by sensor-equipped MECs allows an extended preparation of driving tasks from the MEC to the vehicle. A similarity we seek to exploit further. With the help of a management prototype, we show the feasibility of our approach in principle and how such a system can be realised. A further timing analysis is derived in order to investigate the overhead the proposed management toolchain is introducing.**

***Index Terms*—mobile edge cloud, real-time systems, Kubernetes, automotive systems**

I. Introduction

The current trend towards fully autonomous driving up to SAE Level 5 requires future vehicles to provide a dynamic driving toolchain that is capable of holistically sensing the current traffic scenario, classifying and validating gathered sensory data, and then applying these data points as input to dynamic driving task algorithms [1]. Such toolchains often rely on machine learning (ML) or stochastic algorithms for image classification, object detection, or motion prediction at almost every step [2]. Since these approaches are mostly based on probability distributions, they often pose an inherent problem for the safety of autonomous driving and are therefore deliberately designed to be robust against perturbations. The highly dynamic nature of various traffic situations also places great demands on the available computing capacity and simultaneously inherits this variability of the situation into its computation [2].

As autonomous systems, vehicles suffer from limited computational capabilities, as well as energy, space, and weight constraints, so they are often supported by connecting additional computational resources, such as a mobile edge cloud (MEC) at the roadside [3], [4]. These MECs are often also equipped with sensors on gantries and can thus assess current driving situations at the respective location and prepare maneuver calculations, leveraging the computation demands off the vehicles [3], [5].

In this paper we want to address this situation by exploiting and extending the similarities of vehicle and MEC from a point of view of task reusability and scheduling. This enables an increase in the safety of the system by increasing the predictability of its task behaviour while decreasing the effort on implementation and code / run time analysis.

We will first give an overview of the problem statement and motivation, then go into comparison with related work and show our approach. A prototype illustrating general feasibility of our approach is described and derived timing analysis is provided. Finally we conclude by depicting the achievements and future work to do.

II. Motivation and Problem Statement

Although the MEC is capable of providing real-time driving task functionality to vehicles in its surroundings, like offloading, extending (with gantry sensor data) or adding parts of the dynamic driving toolchain, the introduction of such decoupled functionalities always imply offloading or even transfer costs [6]. In addition to the time delay imposed by offloading, task runtime parameters have to be known in advance. These are required for the vehicle to safely start the offloading procedure without the risk of missing a deadline on one of its offloaded tasks and thus fail the current driving task requirements with potentially critical impacts [6]. Furthermore, sudden connection failures between MEC and vehicle can occur, rendering previously carried out offloading of tasks useless and potentially harmful. We therefore propose a vehicle edge cloud design, where the MEC at roadside is designed as a stationary twin of the vehicle. MEC and vehicle therefore share their applications, operating system and hardware design, which is upscaled (on the MEC side) with the use of Kubernetes containers. The management toolchain proposed in this paper allows for enabling this particular use case and the full self-sufficiency of both vehicle and MEC. Based on this design, the vehicle is provided with full

This research was funded by the Federal Ministry of Transport and Digital Infrastructure of Germany in the research project Providentia++.

autonomy, integrating the MEC as potentially unreliable and less performant ECU clone for real-time task executions into its scheduling algorithms and policy. Furthermore, applications can be developed, tested and verified only once if we assume that they have a modular design and that used algorithms can cope with certain differences of input values (e.g. traffic observation angle). Thus, the MEC can be used as a HIL pre-testing setup for updates or new rollouts of driving functions to the vehicle, in order to ensure correct application functionality. In addition, the use of identical hardware/OS (e.g. ARM, RISC-V / RTOS, RT-enabled Linux) in the MEC and vehicle means that similar system behaviour can be exploited and thus runtime data can be obtained, which in turn can provide clues to the runtime behaviour in the vehicle or, in the best case, even be highly similar. Previous research has shown, that such a scenario might be possible through the use of real-time kernel patches or co-kernel approaches, but also show drawbacks or unsolved challenges [7]. We assume that this approach can be combined with the ML-based deployment and migrations strategy researched in [8]. As of now, we solely focus on the combination and direct connection of MEC and vehicle, as further extensions to central cloud systems imply new limitations in terms of transfer overheads or hardware architectures.

III. Related Work

Previous research on this topic has already shown that it is possible to offload real-time applications [9] or whole workflows [10] in a cloud environment.

In [9], a framework to offload low critical tasks to a cloud environment is presented, where scheduling and offloading decision are supported by machine learning but require an always connected cloud to offload low critical tasks. The work misses some key points of our proposed MEC environment, like the dynamic driving situation and changing vehicle position.

In [11], a platform is designed for seamless deployment and management of container clusters, similar to our approach. They show very quick response times for requests ranging from 3ms to 370ms. Nevertheless, this approach focuses on a centralized cloud infrastructure, where a main cloud is delegating apps to clusters on the edge. A similar approach is shown in [12], where applications for the MEC are provided that help with predictive cloud bursting. However, they again use a centralized architecture and are addressing different usage scenario.

Whereas [13] investigates on the 5G connection and the custom Kubernetes scheduler, which improve latency and load up times, whenever multiple containers are allocated onto the edge server.

A traffic-aware dynamic container migration using LXC containers with real-time kernels on lightweight application containers is considered in [6].

Furthermore, the general idea of real-time containers and their deployment with respect to time guarantees [14], and with a focus on real-time and best effort container co-location inside Kubernetes [15] is just recently getting research interest. The described papers clearly show that offloading of real-time tasks to the MEC is possible and that a Kubernetes driven MEC is capable of hosting real-time applications, even with the means of machine learning-based scheduling. Nevertheless, to the best of our knowledge, there has not yet been an approach that unites a vehicle-centric MEC and a light-weight Kubernetes approach as we describe it.

IV. Approach

We separate the approach into three subsections framing the general idea, the setup based on this idea and possible use case scenarios.

A. General Idea

The overall idea of the proposed approach is that the MEC is a nearly identical hardware and software twin of the vehicle and that it is capable of providing extended RT-services to a vehicle if present in the current area of driving. The MEC is designed to be decentralized and divided into smaller units containing one or more server clusters and gantries equipped with sensors as well as 5G base stations for communication [16] . On top of these clusters, Kubernetes is introduced allowing for automated deployment and scaling, as well as a basic management of containerized real-time applications. In order to fully utilize and adapt the container management to the automotive real-time MEC environment, while also enabling task offloading, we introduce Mark8s, a **M**anagement toolchain for **a**utomotive **r**eal-time Kubernetes containers (**k8s**). It implements a communication gateway per MEC unit enabling fast vehicle-based requests for computational resources within a certain road sector the vehicle is passing. In order to be fully decentralized MEC units are equipped with adjacent neighbour discovery, as well as inter-gateway communication supporting the exchange of health information, such as available resources and the unit's uptime status between two or more adjacent MEC units. Combining the computing capability of the MEC unit's Kubernetes cluster with gantry sensor systems will also allow for offering real-time capable services beyond task offloading from the vehicle or providing pre-computed cloud information. This combination, in addition to a modularization of the automotive driving task chain (sensing, calculating, acting), enables the development and extensive testing of such tasks on the vehicle-like MEC before applying them to the more critical vehicle environment. A permanent sensing container, for example, will fuse and classify found objects for optimal driving pathway calculation before the information is sent to the vehicle and eventually cross-checked with internal sensor and calculation data. Although such driving scenarios and their associated automotive toolchains can vary widely, a common scheme consisting of three kinds of containers can be used as a basis:

- **One-shot containers** (individual/sensitive services) are only used by a single client/vehicle. If the container is no longer needed, gathered data will be deleted and it will be shut down (e.g. offloaded driving tasks)

• **Multi-session containers** (multi-user services) are started once at least one client has requested them. Additional users will simply get redirected to the already running instance. Containers will get shut down, if no client is actively using them after a certain grace period (e.g. driving convoy applications).
• **Permanent containers** (permanent services) start running without being requested and are a special kind of multi-session container, which do not implement a grace period (e.g. automatic emergency detection & alarming services).

Underlining the similarity of MEC and vehicle even further, all examples given could be executed on the vehicle itself, just requiring different sensor data and action receiver modules. Besides other advantages, the containerized applications enable the car manufacturers (OEMs) to keep the whole chain of software components within their development workflows and to guarantee an OEM secured car control when deploying on potentially foreign MEC infrastructure. It also facilitates further follow-up questions on (ethical) responsibility and accounting [17].

B. Setup

The setup derived from the general idea is depicted in figure 1. As mentioned before, the MEC units consist of one ore more gantries and server clusters running Mark8s with Kubernetes. MEC/Mark8s units are connected to the vehicle via low latency 5G radio cells allowing for direct wireless network connection with the prototype gateway at the MEC, where requests are forwarded to the k8s master, which in turn manages different k8s nodes and the containers executed on them.
As the proposed overall architecture of the MEC is decentralized, its MEC units or prototype clusters act autonomously. If requests cannot be handled, the Mark8s cluster gateway will redirect the client to an adjacent prototype gateway/cluster on the predicted future path of the vehicle. Information about such adjacent clusters is asynchronously gathered from a central status aggregation, where newly started prototype clusters report their location and availability after installation. Further health and connectivity status of adjacent clusters is checked bilaterally between neighbouring prototype clusters. Thus, newly installed clusters will report themselves to the global discovery, get neighbouring cluster information and start checking their status and availability. Afterwards, the central status aggregation would only be needed for newly added clusters, or if clusters change their connection parameters and are thus no longer bilaterally detectable. Such unreachable or crashed clusters are no longer used for load balancing or computation considerations but do not affect the availability of adjacent clusters, as long as the deployment/networking infrastructure of each cluster is set up in a self-sufficient manner.

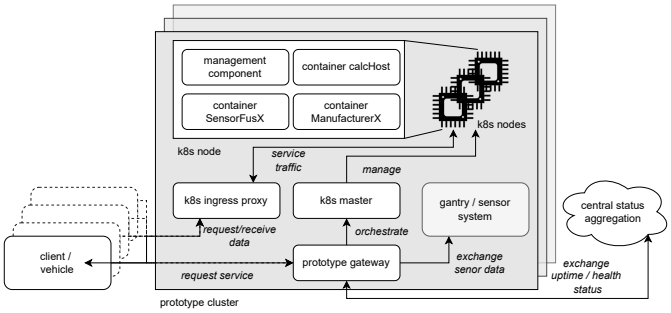

Fig. 1: Sketch of the setup scenario

C. Use Case Scenario

Illustrating the value and functional principle of the setup, we constructed a use case scenario enabled by the prototypical implementation. A vehicle approaching the feeder road of a highway is setting up a connection via 5G network to a MEC/prototype cluster unit capable of task offloading, enhancing or extending. Once the connection is established, an automated request for needed services is sent to the prototype gateway. The gateway will then, based on the workload of the downstream Kubernetes cluster, either schedule the requested service and return the service address after it was successfully launched (SCHEDULED_HERE), redirect it to another gateway/cluster (OTHER_GATEWAY) or deny (CAN_NOT_SCHEDULE) the request.
In the event that the requested service could be scheduled (SCHEDULED_HERE), the responsible prototype cluster is starting up and providing the containerized application, leveraging the computational needs of the vehicle, e.g. preprocessing of a video stream for entertainment or augmented transparency of surrounding vehicles. Other driving-related use cases are also conceivable, such as a predictive lane assistant outsourced to the MEC with increased potential for smoother driving maneuvers and an extended prediction period due to a dramatically expanded field of view at gantry bridges. Figure 2 shows a sequence diagram of a successful client request, which is being scheduled on the local prototype cluster, finishing with the Kubernetes objects being deployed. Once requested, such a service container has to be periodically flagged as still active by the client. This is done because, as the vehicle progresses on its way, it might lose connection or autonomously connect to a following prototype cluster unit and radio cell. On top of these automatically initiated driving tasks, the vehicle can offer additional manually triggered services for passengers, which are supported by the prototype cluster.

V. Demonstrator and Timing Analysis

In our setup two multi-core Cavium ThunderX servers represent the MEC in figure 1 and are running vanilla Ubuntu 20.04. Within this prototype setup all participants are connected via Ethernet. A direct 5G connection is omitted and assumed working as in [18], [16]. To achieve real-time scheduling capabilities, the PREEMPT_RT patched source

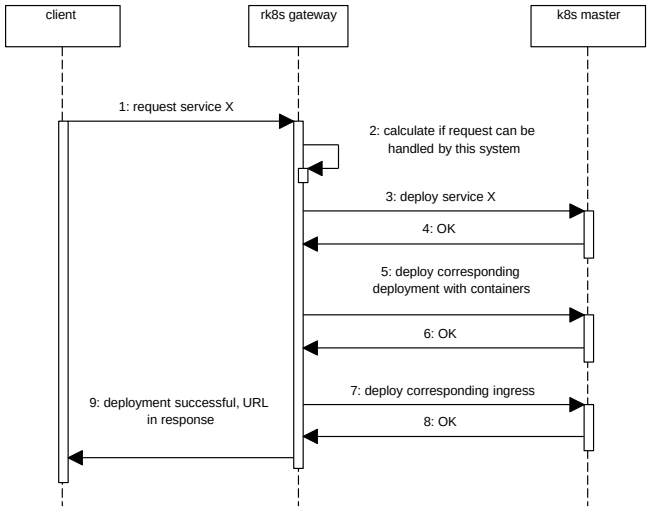

Fig. 2: Sequence diagram of a successful client request

code of the Linux kernel was used and compiled enabling the option Fully Preemptible Kernel. Kubernetes is deployed and runs on top of this real-time environment. The main component of the MarK8s prototype cluster is the gateway, which was designed to be light-weight and is therefore written in node.js for this demonstration. It uses a simple SQLite database to store operating information in it, like deployment service parameters or health status. To exchange data between vehicle and gateway - as well as for other communication means - a REST-API is used.

As previous research has already shown the feasability of a real-time capable Kubernetes as well as the potential of offloading real-time applications (section III), we want to focus on the measurement and analysis of the service request (container upstart and prior decision making) as a main bottle neck of our approach. Runtime and benchmarking tests concerning the real-time capability showed that the PREEMPT_RT was successfully applied to the benchmarking task containers. Measurements upstarting an NGINX (alpine) example native systemd app (4829 ms) and Kubernetes preloaded container (5010 ms) only revealed an overhead of 181 ms in average. Cold starts with containers from a remote source took 8731 ms and multi-session container starts - forwarding the request to a running container - took 85 ms. However, the most important measurement, showing the applicability of the prototype, is the request-response period between a client's request and the answer from the Mark8s prototype gateway. The gateway could either schedule a requested service locally (SCHEDULED_HERE), redirect the client to another gateway (OTHER_GATEWAY) or reject the request as not schedulable at the moment (CAN_NOT_SCHEDULE). No matter what the final result of the request is, the gateway needs to send the responses as fast as possible to enable the client to make an informed decision. Every scheduling decision case was tested 50 times. Additionally we carried out tests for CAN_NOT_SCHEDULE with the global discovery reachable and not reachable to cover the worst case where the MEC unit is isolated. Figure 3 shows the accumulated times measured on average and the division into the different logical code sections. It depicts that SCHEDULED_HERE has the longest response time (272.1 ms), the redirection OTHER_GATEWAY is in the middle (52.25 ms) and the CAN_NOT_SCHEDULE response is the quickest (55,08 ms).

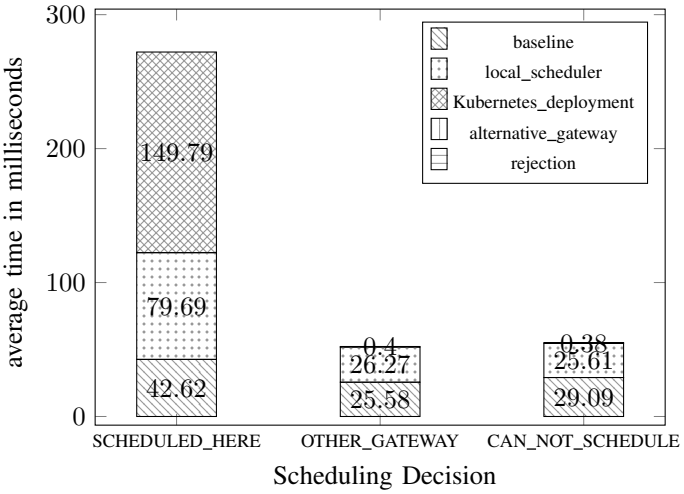

Fig. 3: Request response time diagram for the 3 different response variants following a client's request

Furthermore table I show more detailed numbers and gives further values derived from the measurements. The overall value distribution can be explained by taking a deeper look into the functionality and code structure of the three response options. They all share a baseline part and a local_scheduler part, which checks the schedulability of the request locally. However, if the request is schedulable, more effort has to be done for the local deployment. If it is not schedulable locally, the list of adjacent gateways has to be checked and its address has to be forwarded to the requesting client. If neither is true and there simply is no available adjacent gateway the request can be denied quickly.

Given the above mentioned use case, where a vehicle is entering the MEC-enabled section of a smart road, these first timing analyses show promising results which can only be further evaluated with real workload containers and the full integration of Mark8s with other approaches like [15] and [14].

VI. Conclusion

We propose a **M**anagement approach for **a**utomotive **r**eal-time Kubernetes Containers in the Edge Cloud (Mark8s). The idea of a vehicle-centred automotive future is presented, in which single MEC units are designed as independent hardware and software alike stationary vehicles. The overall idea as well as the design of the toolchain aims to improve fault-tolerance, availability and reusability of applications (modules) from the MEC to the vehicle, while also allowing for rapid development and extensive testing of new driving applications for robustness. The presented prototype shows the feasibility

Scheduling Decision	Phase	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
SCHEDULED_HERE	baseline	26.544	29.117	32.135	42.628	36.942	149.990
	local_scheduler	24.157	28.923	32.562	79.694	39.572	1286.690
	Kubernetes_deployment	99.597	112.066	126.370	149.799	160.977	422.249
OTHER_GATEWAY	baseline	21.412	22.483	24.205	25.587	26.400	58.927
	local_scheduler	21.757	22.770	23.823	26.275	25.462	99.184
	alternative_gateway	0.299	0.340	0.378	0.408	0.410	1.125
CAN_NOT_SCHEDULE with global discovery	baseline	21.414	22.419	23.552	29.094	24.430	134.322
	local_scheduler	20.995	22.787	23.545	25.619	24.968	101.707
	rejection	0.313	0.346	0.367	0.385	0.386	0.879
CAN_NOT_SCHEDULE without global discovery	baseline	19.818	20.905	22.497	24.393	24.278	51.964
	local_scheduler	19.700	21.460	23.277	23.762	24.940	34.613
	rejection	0.270	0.301	0.331	0.359	0.359	1.103

TABLE I: Measurements of Mark8s' scheduling decision timings: SCHEDULED_HERE, OTHER_GATEWAY and CAN_NOT_SCHEDULE with and without working global discovery given in milliseconds

of offloading real-time tasks within such a scenario and gives a first timing analysis in such a system, which can be further enriched by enabling built-in safety and automation features of Kubernetes. Finally, the proposed approach is not very invasive and uses existing software components, is therefore a good basis for future extensions and enhancements, like RT-Kubernetes [14] or REACT [15].

VII. Future Work

While we can show that our approach is feasible for a real-time automotive mobile edge cloud environment, certain restrictions, like privilege escalation due to the used SYS_NICE capability in the prototype, remain. This restriction could be lifted using other available methods like Co-Kernels [7], depending on the use case. Furthermore, useful extensions to the orchestration and managing capabilities of Mark8s, like predictive container start on gateways at the pathway of a vehicle, are not implemented yet. As research on the excluded vehicle part is, as well, still ongoing, we currently try extending the toolchain with a vehicle-centric ML-supported schedulability analysis, as presented in [8].

References

[1] Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles, ISO/SAE PAS 22736 (2021-08), 2021.

[2] E. Yurtsever, J. Lambert, A. Carballo and K. Takeda, "A Survey of Autonomous Driving: Common Practices and Emerging Technologies," in IEEE Access, vol. 8, pp. 58443-58469, 2020, doi: 10.1109/ACCESS.2020.2983149.

[3] S. Raza, S. Wang, M. Ahmed & M.R. Anwar (2019). A Survey on Vehicular Edge Computing: Architecture, Applications, Technical Issues, and Future Directions. Wireless Communications and Mobile Computing, 2019, 3159762. https://doi.org/10.1155/2019/3159762

[4] Intel Corp, "ECU Consolidation Reduces Vehicle Cost, Weight, and Testing", Intel Corp, Accessed: Oct. 27, 2021. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ecu-consolidation-white-paper.pdf

[5] T. Fleck et al. (2018). Towards Large Scale Urban Traffic Reference Data: Smart Infrastructure in the Test Area Autonomous Driving Baden-Württemberg.

[6] S. Maheshwari, S. Choudhury, I. Seskar, and D. Raychaudhuri. "Traffic-Aware Dynamic Container Migration for Real-Time Support in Mobile Edge Clouds." In: 2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS). 2018, pp. 1–6. doi: 10.1109/ANTS.2018.8710163.

[7] V. Struhár, M. Behnam, M. Ashjaei and A. V. Papadopoulos, "Real-Time Containers: A Survey" , 2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020), pp. 7:1–7:9, 2020, doi: 10.4230/OASIcs.Fog-IoT.2020.7

[8] O. Delgadillo, B. Blieninger, J. Kuhn and U. Baumgarten, "A Generalistic Approach to Machine-Learning-Supported Task Migration on Real-Time Systems.", J. Low Power Electron. Appl., 2022, doi:10.3390/jlpea12020026

[9] M.A. Maruf and A. Azim, "Extending resources for avoiding overloads of mixed-criticality tasks in cyber-physical systems", IET Cyber-Physical Systems: Theory & Applications, pp. 60-70., 2020, doi: 10.1049/iet-cps.2018.5062

[10] J. Zhou, J. Sun, M. Zhang and Y. Ma, "Dependable Scheduling for Real-Time Workflows on Cyber–Physical Cloud Systems," in IEEE Transactions on Industrial Informatics, vol. 17, no. 11, pp. 7820-7829, Nov. 2021, doi: 10.1109/TII.2020.3011506.

[11] H. Mfula, A. Ylä-Jääski and J.K. Nurminen, "Seamless Kubernetes Cluster Management in Multi-Cloud and Edge 5G Applications.", In: International Conference on High Performance Computing & Simulation (HPCS 2020). 2021.

[12] F. Faticanti et al. , "Distributed Cloud Intelligence: Implementing an ETSI MANO-Compliant Predictive Cloud Bursting Solution Using Openstack and Kubernetes", In: K. Djemame et al. (eds) Economics of Grids, Clouds, Systems, and Services. GECON 2020. Lecture Notes in Computer Science, vol 12441. Springer, Cham. doi: 10.1007/978-3-030-63058-4_8

[13] M. C. Ogbuachi, A. Reale, P. Suskovics and B. Kovács, "Context-Aware Kubernetes Scheduler for Edge-native Applications on 5G," in Journal of Communications Software and Systems, vol. 16, no. 1, pp. 85-94, April 2020, doi: 10.24138/jcomss.v16i1.1027

[14] S. Fiori, L. Abeni and T. Cucinotta, "RT-kubernetes: containerized real-time cloud computing", In Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing (SAC '22), Association for Computing Machinery, New York, NY, USA, 2022, pp. 36–39, doi:10.1145/3477314.3507216

[15] V. Struhár, S. S. Craciunas, M. Ashjaei, M. Behnam and A. V. Papadopoulos, "REACT: Enabling Real-Time Container Orchestration," 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2021, pp. 1-8, doi: 10.1109/ETFA45728.2021.9613685.

[16] V. Lakshminarasimhan and A. Knoll, "C-V2X Resource Deployment Architecture Based on Moving Network Convoys," 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring), 2020, pp. 1-6, doi: 10.1109/VTC2020-Spring48590.2020.9128410.

[17] J. Gogoll and J.F. Müller, "Autonomous cars: in favor of a mandatory ethics setting" , Science and engineering ethics, 2017, vol. 23 , no. 3, pp. 681-700.

[18] M. Tao, K. Ota and M. Dong, "Foud: Integrating Fog and Cloud for 5G-Enabled V2G Networks," in IEEE Network, vol. 31, no. 2, pp. 8-13, March/April 2017, doi: 10.1109/MNET.2017.1600213NM.

Towards a Predictable and Cognitive Edge-Cloud Architecture for Industrial Systems

Mohammad Ashjaei*, Saad Mubeen*, Masoud Daneshtalab *, Victor Casamayor†, Geoffrey Nelissen‡
*Mälardalen University, Sweden
†Technical University of Vienna, Austria
‡Eindhoven University of Technology, the Netherlands
*firstname.lastname@mdu.se, †v.casamayor@dsg.tuwien.ac.at, ‡g.r.r.j.p.nelissen@tue.nl

***Abstract*—In this paper, we present a conceptual proposal for a novel predictable and cognitive edge-cloud computing architecture for industrial cyber-physical systems. Timing predictability in this multi-layer architecture is envisioned to be supported by cognitive adaptation mechanisms in various computing layers of the edge-cloud computing continuum, including the communication among the layers. We also discuss our preliminary plan to realize the proposed architecture. Furthermore, we conceptualize the proposed architecture on a use case from the automation industry to show its applicability.**

I. Introduction

The edge and cloud computing technologies have already become an integral part of many industrial Cyber Physical Systems (CPS), from infrastructure monitoring, smart automation and construction equipment to telecommunication infrastructures [1], [2]. In such industrial systems, the environment is considered to be highly dynamic. A typical case would be that of construction quarries which are subject to frequent changes in their environment (due to weather, layout modification, etc.) and that accommodate battery-operated construction vehicles that regularly join and leave the site to, for example, fulfill variable charging requests. Beside their requirements for adaptation to varying operating conditions, most of these systems require timing predictable services in various computing layers, e.g., edge and cloud computing layers, as often industrial systems possess strict timing requirements, i.e., a hard service deadline should be met.

The computing continuum, from edge to cloud, that is available today lacks a holistic support for *cognitivity*, *adaptivity* and *timing predictability* in these industrial CPS. In this context, cognitivity refers to processes that monitor the environment, intelligently perceive the situation, and autonomously adapt the overall utilisation of resources, including computation and communication resources. A system is considered timing predictable if it is possible to prove or demonstrate that it meets all the specified timing requirements [3]–[5]. The primary objective of the overall adaptation is to obtain optimal resource utilisation and reduction of overall cost and energy.

To meet these requirements, this paper proposes a novel predictable and cognitive edge-cloud computing architecture for industrial CPS. Timing predictability is supported by cognitive adaptation mechanisms in the edge-cloud architectures. Therefore, we use the term predictable cognitive edge-cloud computing continuum throughout the paper to spotlight on the novel feature of the proposed architecture. That is, timing predictability of services, also known as timeliness of services, will be supported through the use of cognitive and adaptation mechanisms in various layers of the computing continuum. We will leverage AI-enabled technologies that allow the development of computing continuum, from edge to cloud [6]. Achieving such an architecture is of paramount importance to many Original Equipment Manufacturers (OEMs) in their path towards providing not only timing predictable services but also energy- and cost-effective systems that are cooperative and support zero downtime.

II. Related Work

The concurrent execution of an application through all computing continuum layers increases the complexity of the entire system. Thus, holistic approaches have been considered in the literature which tackle the design of the architecture from different aspects, e.g., for mobile models [7], ad-hoc computing establishment [8], and orchestration mechanisms for the edge-cloud computing systems [1]. The architecture proposed in this paper will also focus on utilising AI technologies to allow a seamless integration of adaptive mechanisms into the architecture, while maintaining timeliness of services in all computing layers. Therefore, the proposed architecture considers timeliness aspects, both in computation and communication, unlike the previously proposed architectures, that makes it suitable for time-critical industrial systems.

Considering solely the application orchestration, a comprehensive survey presents various proposals [1]. There are orchestration solutions for cloud computing, which are also evaluated on edge computing, encompassing the edge-cloud computing continuum [9]. One of the key elements of orchestration is the scheduling of services, where few proposals have shown some development in this direction [10]. Predictability in cloud computing has been a focus of several works [11], [12], mainly targeting timeliness for applications in the cloud. A few recent works further extend the predictability for the computing continuum [13].

Focusing on the network technologies in the edge and fog computing, there are very few works that address both timeliness and adaptivity of the communication services. For instance, a self-configuring time-sensitive network (TSN) is proposed for fog and edge computing in the automation domain [14] and for enabling fog computing to use TSN

technologies [15]. In this paper, we will focus on developing communication infrastructure for the edge-cloud computing architecture with the goal of enabling them for utilising predictable communication technologies, such as TSN and wireless TSN, and at the same time providing dynamicity in the configuration of the network.

There are several large EU projects that have initiated the development of edge-cloud computing continuum. For example, the AI@Edge[1] project aims at developing reusable, secure and trustworthy AI solutions for the network edge. The SERRANO[2] project aims at developing an abstraction layer for automated and cognitive orchestration from edge to cloud computing. Development of a set of tools are the main goal in the DITAS[3] project, while Fog-protect[4] targets data protection through the computing continuum. The SESAME[5] project aims at combining solutions of network virtualization with edge computing to develop multi-service 5G small cells to obtain low-latency communication.

To the best of our knowledge, none of the existing architectures particularly present how timeliness and predictability for services can be achieved in various computing layers. Timeliness is a paramount requirement that is imposed by industrial systems as they have many timing requirements to fulfill. Timeliness of services is often left as a secondary objective, while providing computing resources is commonly the primary goal. Therefore, we aim at building an edge-cloud computing continuum that essentially supports predictability in all computing layers including communication among the computing layers. We exploit the concept of cognitivity, including monitoring and adaptation mechanisms, to enable support for predictability of services.

III. Envisioned Architecture

The predictable and cognitive edge-cloud architecture envisioned in this paper is depicted in Fig. 1. In this architecture, the edge-cloud computing continuum consists of several layers of interconnected computing resources. In general, the edge nodes provide services to end-systems or devices, while a cluster of fog nodes aggregate several edge nodes providing enhanced computing capabilities and connectivity. The cloud computing layer, which can consist of several layers itself including enterprise (private) and public clouds, has a vast quantity of computing resources and can steer large computations with massive storage capabilities. The edge node provides services with strict timing requirements, while less time-critical services are deployed to fog or cloud layers. With such an architecture, enterprises can take advantage of multi-layer computing systems for their applications with different requirements on timing.

We aim at leveraging cognitive methodologies in order to support predictable services within the presented multi-layer architecture. This computing architecture can provide services to the end nodes from different computing layers with different requirements. In this work we focus on timing requirements, however, we envision an architecture able to deal with other type of requirements, such as reliability or security requirements. For instance, a service for high time- and safety-critical requirements may only be provided by the fog cluster, while less-critical services can be deployed on the private or even public cloud. The novelty of this architecture is to provide timeliness in all layers, including the private and public cloud, with different time-criticality levels.

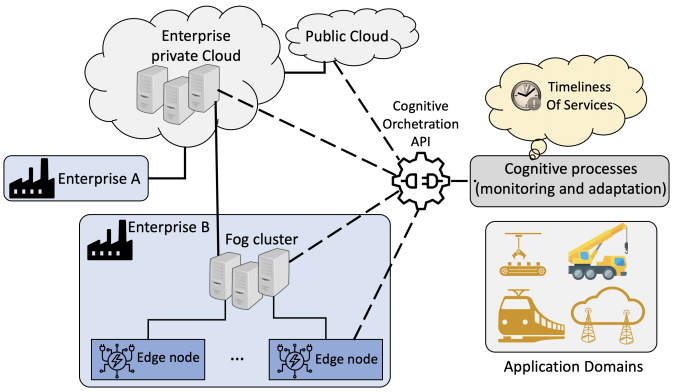

Fig. 1. Envisioned predictable & cognitive edge-cloud computing architecture.

In order to provide such timeliness, during run-time of the system without any service disruption, an entity is envisioned to perform cognitive processes. This entity can reside in different layers adhere to either a centralized, a distributed or a hybrid model. The cognitive orchestration Application Programming Interface (API) provides an interface between the cognitive processes. Moreover, AI and Machine Learning (ML) techniques will be utilized in the cognitive processes to ensure intelligent, autonomous and time-predictable changes during the run-time of the system. The cognitive processes are categorized into three phases: monitoring the environment, intelligent perception of the monitored environment, and on-the-fly adaptation of the computing continuum.

We aim at achieving the following objectives to realize the envisioned architecture:

- to develop a multi-layer edge-cloud computing architecture with an intelligent cognitive capacity to adapt the system autonomously during run-time, while maintaining the timing requirements of the services imposed by industrial systems;
- to adopt suitable AI- and ML-based techniques to perform run-time monitoring, intelligent perception, and adaptation of the multi-layer edge-cloud computing architecture;
- to develop techniques to verify the timing predictability of the system, in terms of computation and communication, during offline and run-time of the system; and
- to demonstrate the proposed architecture together with cognitive processes on realistic or industrial use cases.

IV. Preliminary Plan to Realize the Architecture

The envisioned architecture will be realized as a hierarchical model consisting of several computing layers from edge nodes

[1]https://cordis.europa.eu/project/id/101015922
[2]https://ict-serrano.eu/
[3]https://www.ditas-project.eu/
[4]https://fogprotect.eu/
[5]https://cordis.europa.eu/project/id/671596/results

to a cluster of fog nodes, until the cloud (including enterprise private and public clouds).

A. Predictable run-time environment and communication

The novel essence of this architecture is its support for timing predictability in all layers. Hence, predictable run-time environments, such as real-time operating systems and real-time orchestration systems, will be utilized. In addition, timing predictability will be considered in communication among several computing layers that should also handle high-bandwidth and low-latency communication over wired and wireless networks. For instance, TSN will be considered for wired communication within and between computing layers [16]. Utilizing 5G, as a wireless technology with low-latency and high-bandwidth support, will be considered where wired connection cannot be established. In addition, converged TSN and 5G communication will also be considered [17].

B. Cognitive Orchestration API

We will investigate various solutions to develop an entity to provide capacity for cognition, for instance, a centralized, a distributed or a hybrid model. A centralized model has the advantage of requiring less synchronization, while a distributed model scales better for large systems. In general, edge-cloud computing systems are complex and large, hence a hybrid model might seem a priori a better solution. In any case, these trade-offs will be investigated to select a suitable model to deploy the cognitive entity.

In brief, the cognitive entity will follow the execution and communication of the distributed application auditing its timeliness and proposing adaptive measures. Then, the communication interface between the edge-cloud system and the cognitive entity will be realized by the cognitive orchestration API. The intention is not to implement an entirely new orchestration API, instead to adopt and enhance the existing solutions to provide such an interface, e.g., Kubernetes (like KubeEdge[6]).

Once the cognitive orchestration API is designed, it will be furnished with a set of ML techniques to provide its cognitive behavior.

C. Cognitive Capacity: Monitoring and Dynamic Adaptation

Cognition requires three phases: monitoring the edge-cloud system and its environment; intelligently perceiving the situation; and autonomously adapting the configuration accordingly.

A set of techniques will be used to efficiently monitor the computation and communication utilization. Different parameters will be considered when such monitoring techniques are designed, e.g., periodicity of monitoring, metrics to monitor, and update rates. Open-source tools, such as Prometheus[7], can be adopted to monitor lower-level metrics of the system.

Half-way between monitoring and intelligent perception can be found techniques of predicting monitoring, which by means of ML/AI techniques are able to predict future outcomes of the monitoring module to enhance the system perception. Then, intelligently perceiving the situation is achieved by inferring the current and future edge-cloud system states with respect to its requirements, anticipating possible Service Level Objectives (SLOs) violations. Finally, the cognitive entity selects the most appropriate adaptive mechanism. To do so, a set of techniques based on ML/AI techniques will be adopted to autonomously adapt the edge-cloud system, in terms of computing or communication capacities. For example, a monitoring technique can regularly inspect the network utilisation within fog nodes and an adaptation technique can adjust the routing of traffic based on the gathered (or historical) data. It can also predict potential congestion and recover proactively. Similarly, high-level metrics for software in a computing platform can be monitored, such as the efficiency of the resources used, which can lead to adaptation techniques based on AI that can automatically optimise their deployment.

D. Support for Timing Predictability

We assume that many control applications that will be deployed on edge, fog or cloud require to meet timing requirements such as deadlines, age and reaction constraints [18]. Similarly, delivering information from a remote computing node to an end station should follow different timing constraints. The timing requirements of systems are usually verified during the design phase of the systems and any changes during the system will negate the verification of the timing. In order to continuously support timing predictability of the services in the proposed architecture, online timing verification mechanisms are required. Any changes that the adaptation mechanism proposes as per the cognitive processes should be verified with respect to timing requirements before their deployment. Hence, we will develop analytical and formal techniques to verify timing predictability of services before deployment of proposed changes by the adaptation mechanisms. Such an online verification will be done on both communication and computation to ensure guaranteed timing for the whole system. For instance, pseudo-polynomial response-time analysis techniques for distributed embedded systems and end-to-end resource analysis techniques [19], [20] can be used to verify the timeliness properties of distributed systems with low time-complexity.

V. Conceptualization on an Industrial Use Case

The envisioned architecture can be applied to any application domain with strict and non-strict timing requirements that employs edge-cloud computing for adaptive and predictable systems, including construction vehicles, railway, telecommunication, and many more. In this section, we conceptualize the architecture on a use case from the automation industry to demonstrate its applicability and usability.

The use case comprises an automation assembly line in which a set of machinery collaboratively produces cell phones. The machinery, including punching machine, cutting machine, assembly machine, etc, are connected via a wired network

[6]https://kubeedge.io/en/

[7]https://prometheus.io/docs/introduction/overview/

based on TSN switched Ethernet running Open Platform Communication - Unified Architecture (OPC UA)[8] in the application layer. The Manufacturing Execution System (MES) on the edge is utilized to deploy the orders and make changes in the assembly lines. However, the MES works statically and any changes in the order, communication between the machines, and processes, should be done offline. This can potentially cause some delays and disruption in the production. Therefore, the main idea of this use case is to further develop the automation system to offload some of the intelligent controls to the fog or enterprise cloud, while the changes in the processes and communication among the machinery become automatic and online. This will potentially help the production system to be dynamic in the sense that when different products are ordered the changes in the production system becomes quick and automatic.

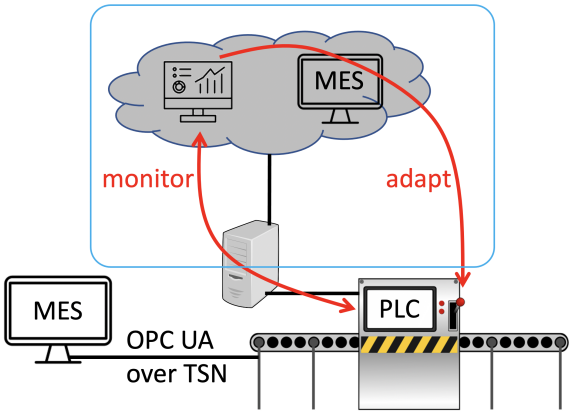

Fig. 2. Conceptualization of the framework on an industrial use case.

We aim to showcase the envisioned architecture and accompanying techniques on this use case, including the edge-cloud computing architecture, cognitive processes to make the system dynamic and automatic, and respect the timing predictability requirements that are imposed by the processes in the assembly line. Within this use case, we plan to use a private in-house cloud to build such an edge-cloud continuum and deploy the MES on both edge and cloud to work collaboratively. Fig. 2 shows the conceptualization of the proposed framework on an existing industrial automation use case.

VI. Conclusion and Future Work

In this paper, we proposed a novel predictable and cognitive edge-cloud computing architecture for industrial CPS. Furthermore, we presented our plan to realize this architecture and develop accompanying techniques. We also provided a conceptualization of the proposed architecture on an industrial automation use case to show its applicability in practice. We believe, the proposed architecture would be beneficial for OEMs in their path towards providing timing predictable and energy- and cost-effective systems that are cooperative and support zero downtime. In the future, we plan to execute the presented plan to realize the proposed architecture.

[8]https://opcfoundation.org/

Acknowledgment

The work in this paper is supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA) via the projects PROVIDENT, DESTINE and INTERCONNECT and by the Swedish Knowledge Foundation via the projects FIESTA, DPAC & HERO.

References

[1] B. Costa, J. Bachiega, L. R. de Carvalho, and A. P. F. Araujo, "Orchestration in fog computing: A comprehensive survey," *ACM Comput. Surv.*, vol. 55, no. 2, 2022.

[2] R. Chaâri, F. Ellouze, A. Koubâa, B. Qureshi, N. Pereira, H. Youssef, and E. Tovar, "Cyber-physical systems clouds: A survey," *Computer Networks*, vol. 108, pp. 260–278, 2016.

[3] J. A. Stankovic and K. Ramamritham, "What is predictability for real-time systems?" *Real-Time Sys.*, vol. 2, no. 4, pp. 247–254, Nov 1990.

[4] D. Grund, J. Reineke, and R. Wilhelm, "A Template for Predictability Definitions with Supporting Evidence," in *Bringing Theory to Practice: Predictability and Performance in Embedded Systems*, ser. OpenAccess Series in Informatics, vol. 18, Dagstuhl, Germany, 2011, pp. 22–31.

[5] S. Mubeen, E. Lisova, and A. Vulgarakis Feljan, "Timing predictability and security in safety-critical industrial cyber-physical systems: A position paper," *Applied Sciences*, vol. 10, no. 9, 2020.

[6] P. Beckman, J. Dongarra, N. Ferrier, G. Fox, T. Moore, D. Reed, and M. Beck, *Harnessing the Computing Continuum for Programming Our World*, 2020, pp. 215–230.

[7] L. Baresi, D. F. Mendonça, M. Garriga, S. Guinea, and G. Quattrocchi, "A unified model for the mobile-edge-cloud continuum," *ACM Trans. Internet Technol.*, vol. 19, no. 2, 2019.

[8] A. M. J. Ferrer, S. Becker, F. Schmidt, L. Thamsen, and O. Kao, "Towards a cognitive compute continuum: An architecture for ad-hoc self-managed swarms," *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pp. 634–641, 2021.

[9] A. Ullah, H. Dagdeviren, R. Ariyattu, J. DesLauriers, T. Kiss, J. Bowden, James, "Micado-edge: Towards an application-level orchestrator for the cloud-to-edge computing continuum," *J. of Grid Comp.*, vol. 19, 12 2021.

[10] G. P. Mattia and R. Beraldi, "Leveraging reinforcement learning for online scheduling of real-time tasks in the edge/fog-to-cloud computing continuum," in *2021 IEEE 20th International Symposium on Network Computing and Applications (NCA)*, 2021.

[11] Y. Gan, Y. Zhang, K. Hu, D. Cheng, Y. He, M. Pancholi, C. Delimitrou, "Leveraging deep learning to improve performance predictability in cloud microservices with seer," *SIGOPS Oper. Syst. Rev.*, vol. 53, no. 1, 2019.

[12] T. Nylander, M. Thelander Andrén, K.-E. Årzén, and M. Maggio, "Cloud application predictability through integrated load-balancing and service time control," in *2018 IEEE International Conference on Autonomic Computing (ICAC)*, 2018, pp. 51–60.

[13] M. Chardet, H. Coullon, and C. Perez, "Predictable efficiency for reconfiguration of service-oriented systems with concerto," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, 2020, pp. 340–349.

[14] M. Gutiérrez, A. Ademaj, W. Steiner, R. Dobrin, and S. Punnekkat, "Self-configuration of ieee 802.1 tsn networks," in *IEEE International Conference on Emerging Technologies and Factory Automation*, 2017.

[15] P. Pop, M. L. Raagaard, M. Gutierrez, and W. Steiner, "Enabling Fog Computing for Industrial Automation Through Time-Sensitive Networking (TSN)," *IEEE Comm. Standards Magazine*, vol. 2, no. 2, 2018.

[16] M. Ashjaei, L. Lo Bello, M. Daneshtalab, G. Patti, S. Saponara, and S. Mubeen, "Time-sensitive networking in automotive embedded systems: State of the art and research opportunities," *Journal of Systems Architecture*, vol. 117, p. 102137, 2021.

[17] Z. Satka, D. Pantzar, A. Magnusson, M. Ashjaei, H. Fotouhi, M. Sjödin, M. Daneshtalab, and S. Mubeen, "Developing a Translation Technique for Converged TSN-5G Communication," in *18th IEEE International Conference on Factory Communication Systems*, 2022.

[18] S. Mubeen, T. Nolte, M. Sjödin, J. Lundbäck, and K.-L. Lundbäck, "Supporting timing analysis of vehicular embedded systems through the refinement of timing constraints," *Software & Systems Modeling*, vol. 18, pp. 39–69, 2019.

[19] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study," *Computer Science and Information Systems*, vol. 10, no. 1, 2013.

[20] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *Journal of Systems Architecture*, vol. 80, pp. 104 – 113, 2017.

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 resources 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 the real-time behavior but also resource usage 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.

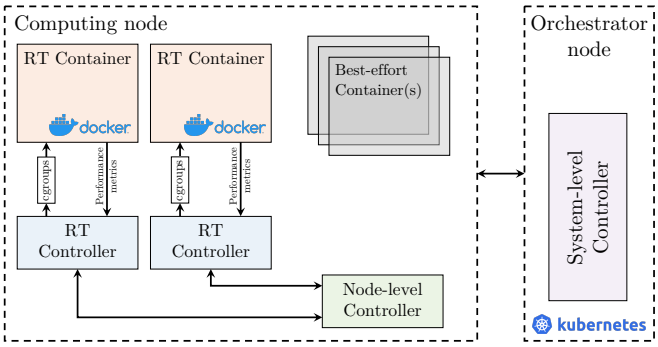

Fig. 1. Overview of the system.

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 OLM 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 to switch 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

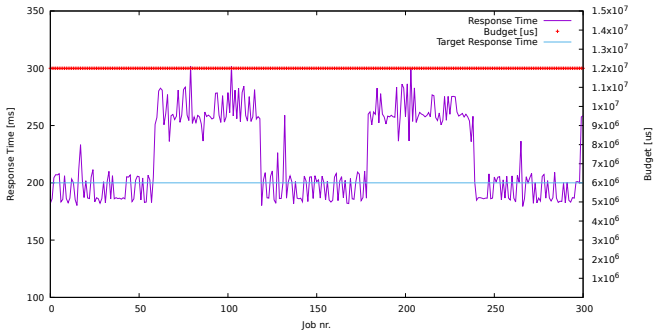

Fig. 2. Response time of a dynamically changing containerized task without online adaptation.

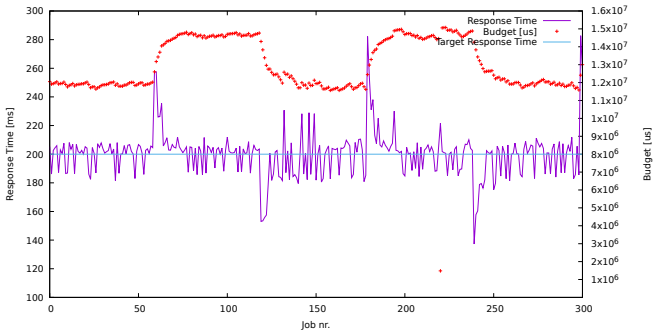

Fig. 3. Response time of a dynamically changing containerized task with online adaptation.

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.

High-performance real-time systems design from cloud to embedded edge

Matteo Maria Andreozzi
Arm
Cambridge, United Kingdom
matteo.andreozzi@arm.com

Girish Shirasat
Arm
Cambridge, United Kingdom
girish.shirasat@arm.com

***Abstract*—Real-time computer systems are rapidly evolving into high-performance heterogeneous systems where co-location of multiple workloads can improve utilisation and re-use of system resources. This, however, comes at the cost of performance degradation due to interference on shared resources, and increased uncertainty. Resource sharing critically increases the need for predictively and deterministically managing the systems' resources. This will become a crucial property of future computing systems following the cloud-native design paradigm, in order to predict worst-case execution times (WCET) for their dynamic real-time workloads before their deployment to the embedded edge. In this work, we'll cover the impact of interference on shared resources in heterogeneous compute platforms, and we'll define the Arm terminology and the principles for high performance real-time. We will also cover system software architectures that are being envisioned in initiatives such as SOAFEE (Scalable Open Architecture for Embedded Edge [2]) to address the need to enable mixed critical workloads and the orchestration of it from cloud to embedded edge.**

***Index Terms*—real-time, Arm, high-performance, mixed-criticality, QoS, SOAFEE**

I. Introduction

In computer-science, a radical shift towards heterogeneous compute platforms is happening now, accelerated by the rise of Machine Learning and thus dedicated accelerators, and the plateauing of the Moore's law applied to CPU compute power. In the real-time computing landscape, this shift has given rise to high-performance real-time applications. On high-performing, heterogeneous systems co-location of multiple mixed criticality workloads on the same SoC can dramatically improve the utilisation of system resources, enabling resource sharing (e.g., IO devices, hardware accelerators, etc.) and improving the efficiency of data sharing across workloads.

However, co-location also comes at the cost of potential performance degradation, both average and worst-case, due to interference on shared resources, and increased uncertainty in terms of workload execution time. Both the academia and industry have been investigating the impact of shared resource contention on real-time and mixed critical software, on hardware requestors (e.g., CPU, GPU, other hardware accelerators) and on memory bandwidth availability, resources access latency, and jitter [12], [13], [14]. The advent of larger integrated platforms which will run real-time workloads alongside GPOS workloads now calls for those systems to being able to provision their resources in a quantifiable and predictable way. This becomes crucial to determine acceptable worst-case execution times (WCET) for real-time workloads and to ensure smooth and responsive operation of the GPOS workloads running alongside them.

To aid compartmentalise traffic streams on shared resources, silicon hardware designers and manufacturers have introduced, primarily in the infrastructure market, technology that allows memory transactions to be labelled and then subsequently confined to partitions of shared resources: Arm, MPAM [3], and Intel, CAT [7] . In this paper, we introduce our key design principles, methodology and metrics for designing high-performance real-time systems. We will also look at the role software plays in achieving such systems.

II. Arm High-Performance Real-Time Design Principles

This section describes the foundation concepts and theory behind designing Arm-based high-performance real-time systems.

A. Real-Time Performance Metrics

Power consumption, performance (typically average or peak performance), and chip area are widely utilized design metrics considered when designing a computing system. Such metrics are typically obtained through measurement under a set of conditions representative of the intended system production deployment operations (platform target workloads). When designing real-time systems, additional performance metrics should be considered, such as quantifying how much the system allows confident computation of worst-case execution times (WCET) for each of the real-time workloads it is being designed to execute [1]. Typically, the degree of uncertainty on computing the WCET that characterizes current high-performance real-time compute platforms makes classical methods of computing the WCET unfeasible (such as analytical) [6]. We therefore advocate the adoption of the following empirical performance metrics: i) Worst-case measured performance and ii) Time-predictability, defined as the quotient between the best-case measured performance and the worst-case measured performance.

B. Sources of uncertainty

The reason for high uncertainty in determining the WCET is typically down to specific sources of uncertainty. The sources

of uncertainty we consider in the following affect the ability to predict or even precisely measure the timing characteristics of real-time systems:

- Workload input data or events: they cause uncertainty when influencing the software control flow or the amount of computation performed by it. In this case we say that the workload is data-variant. For example, conditional branches based on values provided by or calculated from input data can lead to different paths of execution. Also the depth of loops or recursions may depend on the content or size of the input data.
- Hardware state: state of the hardware resources at beginning of execution. Examples are initial cache contents or memory controller row buffer content.
- Interference: deviation in performance caused by workloads that contend for the same shared resources, alter the initial hardware state for other workloads or both.

C. Shared resources and interference channels

As interference arises from contention between workloads, on accessing or using shared resources, co-location of workloads on high-performance system is prone to be affected by such contention, which calls for its accurate quantification. Each hardware shared resource can exhibit one or more interference channel, each one corresponding to a place in the resource where a specific type of contention can happen. The following are examples of potential resource interference channels:

- Internal hardware buffers between pipeline stages: a congested buffer may result in a general resource stall, delaying the service provided by the resource.
- Arbitration policies: they govern which workload has access to the resource at any given time. Biased policies (e.g., strict priority ones) or generally non-work-conserving ones can cause starvation of workload request flows

III. QUALITY OF SERVICE (QOS)

Solutions that address the need for predictively and deterministically managing shared resources are collectively named Quality of Service (QoS). We define here the **QoS principles** for architecting and designing a QoS-enabled computing system capable of delivering differential performance treatment to its users (workloads).

Principle 0 - There is no controllability of a system without observability.

A system where QoS controls are successfully deployed should provide a consistent monitoring infrastructure which can sample the system and provide feedback on the functioning of such controls. It also allows and enable software to discover which shared resources are utilised by a workload.

Principle 1 – QoS is a system-level feature:

A QoS-enabled system should orchestrate its resources in a consistent way, so that the system's users (workloads) are provisioned with certain Levels of Service (memory access bandwidth and latency, compute time, peripheral access, etc.) consistent throughout.

Principle 2 - QoS is quantifiable and predictable:

The set of guarantees a QoS-enabled system can provide should be clear and the level of service that the system will guarantee to its user should be predictable based on system configuration and other conditions.

These principles should be considered both for individual hardware components design and when approaching the whole system design and integration, including software stack and functions. [4]

IV. APPLICATION MODEL

In high-performance real-time systems, hardware should be configurable and configured to provide service guarantees to a certain mix of software, and software should be enabled to manage and monitor the resources of known hardware. We here adopt a workload analysis process, the data-flow model [8], aimed at identifying the QoS requirements of the applications to be deployed on a target system. The data-flow model allows to identify the resources (processing nodes and data paths) which are involved in the execution of such workloads. Those resources will be the ones requiring their service to be characterised, and resource management when contended upon.

We start with capturing use cases requirements, as this is fundamental to enable correct resource provisioning. Workloads should have specific goals. A well-characterised workload is one for which we can specify its QoS requirements and identify a range of QoS values over which it can operate and meet said goals. A generic QoS Framework can manage different types of guarantees, all contributing to various workloads' goals, for instance performance, power, or precision. In the following, we look at leveraging QoS controls to enforce real-time guarantees, i.e., those referring to the timing properties of a real-time workload.

Realtime workloads are typically composed of a collection of entities (program code, devices, data streams) that co-operate in a non-trivial way. Workloads might have elements and functions which are dependent on external events or input, computations which might be triggered dynamically, and which might be unpredictable both in terms of activation time and duration. [5] For such systems, it is generically unfeasible to statically compute any concept such as "anticipated peak or average load" by means of classic real-time compute approaches such as static computation graph analysis. Therefore, to define key timing parameters and constraints, we break workloads down by means of the Data-Flow model.

The Data-Flow model we adopt consists of ***Processing Nodes***, compute elements which react to events, that can produce and/or receive data, and can be either software (e.g., threads) or hardware entities (peripherals), and ***Data Paths***, which can either be physical links such as network interconnections or software communication structures, and enable connecting Processing Nodes to provide their service to the

workload, but are not directly providing it to the workload itself.

According to Bikash S., et. Al [8] we can define QoS as resource management of the end-to-end allocation and scheduling of resources to workloads, based on their QoS requirements, such as:

- Each processing node **N** satisfies its local constraints
- Each data path **P** satisfies the timing constraints of all nodes N it connects, when activated

The mapping of use-cases onto the system's shared resources **N – processing nodes** and **P – data paths**, leads to the identification of interference channels where monitoring and control is needed to preserve data-flow isolation for such use-cases. This is easily identified as the set of N, P which appear in more than one dataflow.

A system capable of delivering high-performance real-time, in which QoS-based resource management is implemented to mitigate interference on its shared resources, should define its following characteristics: Granularity, Resource Monitoring and Resource Control.

A. Granularity

A system providing real-time can do so at different levels of coarseness with respect to how it identifies the users of its resources, i.e., the system **granularity**. A system's **granularity** is defined as the finest resolution at which the system – as whole – can identify users of its shared resources (the set of N and P), both for monitoring and control purposes.

B. Resource Monitoring

Monitoring of data paths and processing nodes provides insight into which shared resources are utilised by workloads, by what extent, and what causes interference on them. It also supports the implementation of control loops in the system by providing feedback to software to adapt its schedule or operate resource controls. A monitoring infrastructure should enable observation of all data paths and processing nodes involved in the computation of the system workloads. A real-time system can provide, for each identified path P, monitoring capabilities to observe performance characteristics that are specific to each shared resource, for instance:

- **Path traversal latency**: the end-to-end latency of the path from input to output. Could be punctual, or aggregated over a time window, with standard deviation and average jitter over the same time window.
- **Path utilization**: the amount of path capacity in use at any given time.
- **Path bandwidth**: i.e., amount of information transferred over the path in a time window.

For each identified processing node N, the monitoring infrastructure can also provide:

- **Node utilization**: information processed/time unit as a proportion of the node maximum capacity
- **Node access latency**: the wait time a user experiences before obtaining compute service from the node, either punctual or aggregated as average over a time window, standard deviation, and average jitter over the same time window

C. Monitor Characterization

Monitors can be characterized in terms of invasiveness, precision, frequency, measurement lag:

- **Invasiveness**: expressed as absolute delta or tolerance variance on the observed values due to perturbation caused by the monitor on the observed metric
- **Precision (resolution)**: expressed as minimum detectable unit of measurement
- **Frequency**: maximum measurement collection frequency (max between sampling frequency and interrogation frequency) at which fresh measurement (not replicas of past values) can be obtained from the monitor
- **Measurement Lag**: maximum measurement collection delay (max between sampling collection and availability to interrogation)

D. Resource Control

A Resource Control infrastructure should enable consistent regulation of all data paths P and processing nodes N identified in the data-flow analysis of the system workloads. Examples of resource controls are:

- **Path traversal latency control**: maximum input (access) to output (exit) latency for the path
- **Path bandwidth control**: minimum amount of data guaranteed transferrable by a resource user over a defined time window
- **Path utilization control**: minimum amount of resource guaranteed to a resource user, expressed as a proportion of the total available resource.

E. Resource Control Characterisation

Similarly, controls can also be characterised in terms of transitory, precision and frequency:

- **Transitory**: maximum time over which the control converges to a new steady state (regulates as intended) after receiving input. Measured as time difference between the time a new input is submitted to the control to the time the control produces a stable and updated output
- **Precision (resolution)**: minimum configurable unit of control
- **Frequency**: maximum number of (re)configurations per unit of time

F. Shared Resources Characterisation

Real-time computing systems are designed to execute workloads where data processed by computing nodes (N) and data flowing between them through paths (P) requires service guarantees. Some form of arbitration - implicit or explicit – will regulate how users are granted access to those shared resources. The execution time of a workload depends upon the forward progress of the shared resourced used by its data flow. The forward progress of a shared resource depends on

the arbitration points they contain. Arbitration points can be managed by Resource Controls. Characterisation of the shared resources consists in specifying what level of service those resources provide to their users based on their arbitration policies and configuration.

For example, the above could be about

- Resource-specific attributes such as local scheduling policies or resource access rules, including configuration options and effects on the policies.
- Resource concurrent access and interference properties

Resource type and performance characteristics, scheduling policies, cost/performance functions will all contribute to resource-specific characterisation. A resource can be characterised when its service curve is known, e.g., when, given a user resource access pattern, its level of service can be predictably determined [1]. This means given a specific input into the resource, and a set configuration for its operable controls, it is possible to know a-priori what will be the measurable outcome of key resource metrics.

V. Standard Hardware Resource Monitors and Controls

The Armv8.4-A Memory System Resource Partitioning and Monitoring (MPAM) extension of the Arm Architecture define mechanisms to provide traffic flows identification throughout the system, and monitoring and control interfaces for MPAM-enabled system resources. MPAM enhances the system memory request and responses with identifiers (PARTID and PMG)

- Partition Identifiers (PARTID) that identify the flow that generated a particular request for the purpose of monitoring and control
- Performance Monitoring Group (PMG) property of PARTIDs, which can be used for the purpose of finer grain monitoring

MPAM enables operating systems or other software entities to assign a PARTID to parts of a workload, and to monitor and control their usage of the MPAM-enabled system shared resources.

A. Monitoring Interfaces

MPAM provides two standard monitoring interfaces, both of which are optional:

- Cache-storage usage monitors that report the cache utilisation for a given PARTID and PMG
- Memory-bandwidth usage monitors that report the number of bytes transferred for a given PARTID and PMG

Monitors can be configured to filter requests by type, for example read or write, and by a choice of PARTID and PMG or PARTID only.

B. Control Interfaces

MPAM provides 6 types of standard control interfaces, all of which are optional:

- Cache-portion partitioning
- Cache maximum-capacity partitioning
- Memory-bandwidth portion partitioning
- Memory-bandwidth minimum and maximum partitioning
- Memory-bandwidth proportional-stride partitioning
- Priority partitioning

Cache-portion partitioning subdivides a cache resource into several portions of equal and fixed size. Cache maximum-capacity partitioning limits the ability of a flow to occupy more than a configurable fraction of the cache capacity. Cache maximum-capacity partitioning can be combined with cache-portion partitioning, for example to restrict the ability of a single flow to occupy all the capacity of cache portions that have been made available to multiple flows.

Memory-bandwidth portion partitioning subdivides memory bandwidth into several portions (quanta). Memory-bandwidth minimum and maximum partitioning allow setting of a minimum guaranteed and maximum permitted memory bandwidth that is applied to a flow in the presence of contention. Memory-bandwidth proportional-stride partitioning is based on a configurable stride for each flow, permitting a flow to consume bandwidth in proportion to its own stride relative to the strides of other flows that are competing for bandwidth.

Priority partitioning provides a way for resources to expose partition-based configuration of internal arbitration policies. These can be used by system software for fine-grained control over scheduling and arbitration policies in the memory system.

VI. CLOUD NATIVE SOFTWARE ARCHITECTURES FOR MIXED CRITICAL WORKLOADS - SOAFEE

In the software defined embedded systems of the future, cloud-native design patterns are being considered to enable an agile DevOps environment for accelerated software feature deployments and increased developer efficiency [11]. As complex system designs get enabled with the resource controls to provide a defined QoS required by an application workload, the cloud native system software architectures and corresponding infrastructure needs to provide the ability to express the workload's spatial and temporal requirements , configure the system to achieve these requirements which includes configuring the the resource controls described in this paper and orchestrate them across the distributed embedded-compute system best suited to achieve those requirements. This is a complex problem to solve because the existing cloud-native infrastructure does not cater to mixed critical workload development and there are no standards-based configuration model or solution for mixed critical workload orchestration.

To enable complex features like mixed critical workload orchestration, the QoS features described in this paper need to be advertised in a standardized way from hardware to firmware through constructs like ACPI/DT and then feed into the operating system before exposing the platform capabilities to an orchestrator like Kubernetes. Once the scheduler in the orchestrator is aware of the system capabilities that includes the general compute attributes like CPU core count, frequency, amount of RAM, etc., along with real-time hardware capabilities like MPAM, it should then configure the system attributes using standardized interfaces and data models to satisfy the

application required service level agreements before deploying the workload into the most appropriate compute element in the system.

There are several technology issues that need to be solved to address the above orchestrator usage scenario. We are listing a few below, including:

- Standardized firmware interfaces for real-time system features from hardware to firmware to operating system.
- Standardized software interfaces from OS to automotive middleware and orchestrators
- Virtual development environment in cloud to enable mixed critical workload development and deployment in production system [10]
- Rich ecosystem of commercially supported and where appropriate functionally safe / certified software components.

The SOAFEE SIG was launched to bring together major ecosystem players, including OEMs, Tier 1s, CSPs, OSVs and ISVs, SIPs and other technology providers to address some of these complex infrastructure issues. SOAFEE SIG will deliver a cloud native architecture that is enhanced for mixed-critical automotive applications and an open-source reference implementation that enables commercial and non-commercial offerings. [2].

VII. CONCLUSIONS

The paper presented an overview of our approach to designing Arm-based systems for high-performance real-time, with attention to workload decomposition and system-level requirements. We have also briefly summarized the Arm architectural support and the complementary software initiative SOAFEE. In follow-up contributions we will expand on our real-time verification methodology and on details pertaining hardware and software support for mixed criticality real-time workloads.

VIII. ACKNOWLEDGMENTS

We would like to acknowledge here the contributions of: Frances Conboy, Sam Danyo, Adrian Herrera, Jan-Peter Larsson, and Andriani Mappoura. We also thank the University of Pisa research group led by Prof. Giovanni Stea for their continued and valued research collaboration relationship with our team.

REFERENCES

[1] Alan Burns and Robert I. Davis. 2017. A Survey of Research into Mixed Criticality Systems. ACM Comput. Surv. 50, 6, Article 82 (November 2018), 37 pages. DOI:https://doi.org/10.1145/3131347

[2] SOAFEE Initiative, http://soafee.io

[3] Arm® Architecture Reference Manual Supplement Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A, available online at https://developer.arm.com/docs/ddi0598/latest

[4] F. Rehm et al., "The Road towards Predictable Automotive High - Performance Platforms," 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2021, pp. 1915-1924, doi: 10.23919/DATE51398.2021.9473996.

[5] M. Andreozzi et al., "Heterogeneous Systems Modelling with Adaptive Traffic Profiles and Its Application to Worst-Case Analysis of a DRAM Controller," 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC), 2020, pp. 79-86, doi: 10.1109/COMPSAC48688.2020.00020.

[6] Marco Paolieri et al., 2009. Hardware support for WCET analysis of hard real-time multicore systems. In Proceedings of the 36th annual international symposium on Computer architecture (ISCA '09). Association for Computing Machinery, New York, NY, USA, 57–68. DOI:https://doi.org/10.1145/1555754.1555764

[7] Intel Cache Allocation Technology, https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology

[8] B. Sabata et al, "Taxomomy of QoS Specifications," in Object-Oriented Real-Time Dependable Systems, IEEE International Workshop on, Newport Beach, CA, 1997 pp. 100. doi: 10.1109/WORDS.1997.609931 url: https://doi.ieeecomputersociety.org/10.1109/WORDS.1997.609931

[9] Matteo Andreozzi et al., A MILP approach to DRAM access worst-case analysis, Computers & Operations Research, Volume 143, 2022, 105774, ISSN 0305-0548, https://doi.org/10.1016/j.cor.2022.105774.

[10] Girish Shirasat et al., "Accelerating Software-Defined Vehicles through Cloud-To-Vehicle Edge Environmental Parity" , https://soafee.io/blog/2022/sdv_with_cloud/

[11] Girish Shirasat "Cloud Native Approach to the Software Defined Car", https://community.arm.com/arm-community-blogs/b/embedded-blog/posts/cloud-native-approach-to-the-software-defined-car

[12] Mohamed Hassan and Rodolfo Pellizzoni. "Bounding DRAM interference in COTS heterogeneous MPSoCs for mixed criticality systems". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37(11):2323–2336, 2018.

[13] F. Restuccia, M. Pagani, A. Biondi, M. Marinoni, and G. Buttazzo. "Modeling and analysis of bus contention for hardware accelerators in FPGA SoCs." 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[14] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo. "A holistic memory contention analysis for parallel real-time tasks under partitioned scheduling". In Proceedings of the 26th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2020), 2020.

Priority-Driven Real-Time Scheduling in ROS 2: Potential and Challenges

Hyunjong Choi, Daniel Enright, Hoora Sobhani, Yecheng Xiang, and Hyoseung Kim
University of California, Riverside
{hchoi036, denri006, hsobh002, yxian013, hyoseung}@ucr.edu

***Abstract*—To ensure timely and safe operations of robotic applications in a highly dynamic and uncertain environment, predictable end-to-end behavior of systems is essential. Although ROS (Robot Operating System) is one of the most prevalent robotic middleware frameworks, it has shown limitations in real-time support over the past decade. With this paper, we argue that the real-time performance and predictability of ROS can be significantly improved by enabling priority-driven scheduling in the framework. To support this argument, we first review our recent work on priority-driven chain-aware scheduling and evaluate it with real-world scenarios through integration into the open-source *reference system*, which was developed by Apex.AI for ROS 2 executor benchmarking. Experimental results on a resource-constrained platform, i.e., Raspberry Pi 4, demonstrate that priority-driven scheduling outperforms the current ROS 2 default scheduling scheme in terms of various key performance indicators, e.g., latency, message drop, and jitter. In addition, we discuss two other challenges, multi-threaded executor design and accelerator support, which have not yet been studied but are essential for better real-time performance in ROS 2.**

I. Introduction

ROS (Robot Operating System) has gained the spotlight among developers in the robotics community by facilitating software modularity and composability in the development of robotic applications. However, over the past decade, ROS has shown major shortcomings in real-time support required for safety-critical applications. Although ROS 2, the new version of ROS, aims to better support real-time capabilities by employing a new software architecture and the Data Distribution Service (DDS), it still remains challenging to guarantee stringent timing constraints in ROS-based systems.

Ensuring predictable end-to-end latency is crucial for applications in a safety-critical domain [7]. However, meeting this requirement in a practical framework like ROS 2 is not a trivial problem due to the following reasons. First, robotic applications generally form a set of processing chains whose data and temporal dependencies are hard to analyze. Second, ROS 2 has a complex and unique scheduling behavior caused by multiple schedulable entities, e.g, callbacks, nodes, and executors, across various abstraction layers, making it difficult to apply existing real-time techniques. Lastly, due to the open-source nature of ROS, many programmers independently develop software components interacting with each other; hence, it is hard to integrate them into a system for a given mission while satisfying their performance requirements.

Research on real-time ROS 2 processing chains has started only recently. Casini et al. [6] proposed a pioneering analysis technique to upper bound the response time of processing chains by modeling ROS executors with resource reservations. Tobias et al. [5] presented enhanced analysis to offer tighter bounds. These studies laid the groundwork to analyze systems developed using ROS 2. On the other hand, we took a completely different approach. While previous studies focused on the unmodified, default ROS 2 scheduling scheme, we found that major limitations, such as long end-to-end latency and pessimism in the analysis, are due to the poor support of prioritization in the existing scheduling scheme. As a result, we developed PiCAS [8], a priority-driven chain-aware scheduling framework for ROS 2, to improve the end-to-end latency of processing chains with predictable bounds. PiCAS also answers how to allocate resources to further improve responsiveness of critical chains.

While some real-time challenges have been studied as discussed above, there are still many open problems that need to be explored for ROS 2. All previous work on ROS 2 processing chains, including our own, only assume single-threaded executors. However, as we will show in Fig. 3, multi-threaded executors have the potential to offer better latency and higher throughput in a system equipped with multiple CPU cores. Besides, since ROS 2 is being widely used in the development of intelligent autonomous systems with machine learning algorithms, unpredictable timing behavior could appear from shared hardware accelerators such as GPU and FPGA, which would be a serious problem in resource-constrained embedded robotic platforms.

In this paper, we will explore the potential of the priority-driven scheduling approach to improve the real-time performance and predictability of ROS 2. We will first review our prior work, PiCAS, and then evaluate it on Raspberry Pi 4 with the reference system [3], which was developed by Apex.AI to benchmark the performance of ROS 2 executors under real-world autonomous driving scenarios. Then we will discuss the two open problems that we are currently working on, i.e., multi-threaded executor design and real-time GPU acceleration support, which would be essential for ROS 2 to serve as a practical yet reliable real-time software infrastructure.

II. Priority-Driven Chain-Aware Scheduling

A. Background

Our priority-driven chain-aware scheduling, PiCAS [8], was motivated by the two major problems of the current ROS 2 framework. First, ROS 2 consists of multiple layers of

abstractions that are not aware of the criticality of processing chains. The unique scheduling behavior of an executor, which schedules timer callbacks always first, makes other callbacks' priorities ineffective. Hence, the current ROS 2 executor ignores the urgency of task chains and results in a fairness-oriented scheduling behavior. Second, the current ROS 2 framework lacks systematic support for resource allocation and latency analysis. This causes poor resource utilization and non-deterministic end-to-end behavior.

To solve these issues, PiCAS enables prioritization of critical computation chains across complex abstraction layers of ROS 2 (see Fig. 1 for overview). We re-designed the current ROS 2 scheduling architecture with the following considerations: (1) higher-priority chain should execute earlier than lower-priority chains, and (2) if the instances of the same chain are assigned to the same CPU core, they should execute in their arrival order. The latter is to reduce self-interference between instances of the same chain, thereby preventing undesirable latency increases.

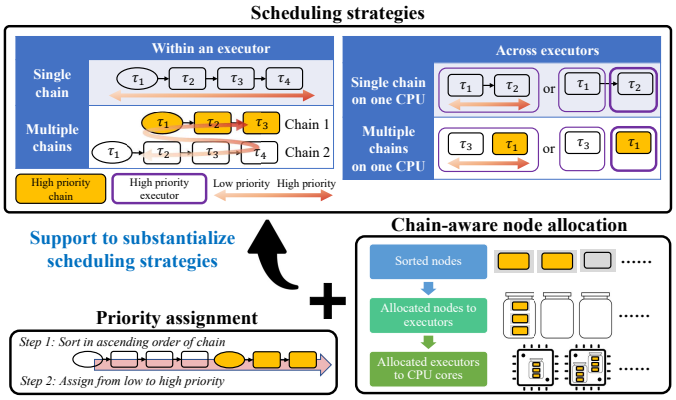

Fig. 1: PiCAS framework

Based on the above considerations, we developed chain scheduling strategies within an executor and across executors. To realize these scheduling strategies, PiCAS introduces a callback priority assignment scheme. It assigns strictly higher priority to callbacks of more critical chains, and within each chain, it prioritizes callbacks in the front to those in the back to avoid the self interference problem. PiCAS also includes a chain-aware node allocation algorithm to allocate given nodes to executors, and then maps executors to available CPU cores while following the scheduling strategies. This algorithm tries to allocate all nodes associated with the same chain to the same CPU core whenever possible in order to minimize interference between different chains.

B. Evaluation of Priority-Driven Scheduling

To understand the benefit of the priority-based scheduling approach under a realistic scenario, we evaluate PiCAS w the reference system [3] that was developed by Apex and introduced at the ROS 2 Real-Time Executor Worksl held in conjunction with ROSCon 2021 [2]. The reference system resembles the lidar-based perception pipeline of Auto-ware.Auto [4], as illustrated in Fig. 2. We integrated PiCAS

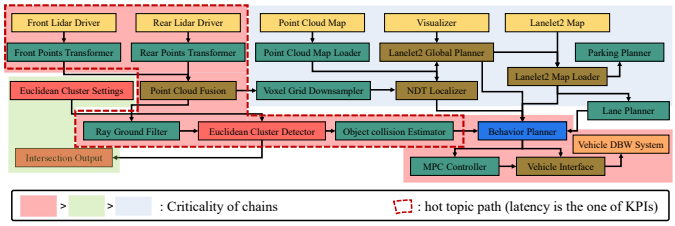

Fig. 2: Chain configuration of Autoware model

into the reference system running on the Galactic version of ROS 2 in a Raspberry Pi 4 platform.

Key performance indicators (KPIs). For ease of benchmarking, the reference system evaluates various KPIs such as:

- *Latency of hot topic path*: In a real-world scenario, the reference system should recognize obstacles as quickly as possible to avoid collisions. Thus, the lower latency from the Front Lidar to the Object Collision Estimator (the red dotted line shape in Fig 2) is better.
- *Number of dropped messages*: Since old sensor data is less valuable than newly sensed data, the old ones can be dropped in favor of the newest sample, but at the cost of information is lost. Therefore, the fewer number of dropped messages is better.
- *Timing jitter of Behavior Planner*: The Behavior Planner node should execute periodically, as accurate as possible according to its set frequency (100 msec). Thus, the lower jitter and drift of this node are better.

Comparison of approaches. We compare the priority-driven scheduling approach (ROS2-PiCAS) with the default ROS 2 scheduling scheme (ROS2-default). Two different executor configurations are considered for ROS2-default: single-threaded and multi-threaded executors. With the single-threaded executor, all nodes are allocated to one thread running on a single CPU core. So, we compare this to PiCAS with one single-threaded executor. The multi-threaded executor runs with as many worker threads as the number of available CPU cores. Since PiCAS does not currently support multi-threaded executors, we use multiple single-threaded executors for PiCAS, i.e., four single-threaded executors on four cores of Raspberry Pi 4, and compare this with the multi-threaded executor of the default ROS 2.

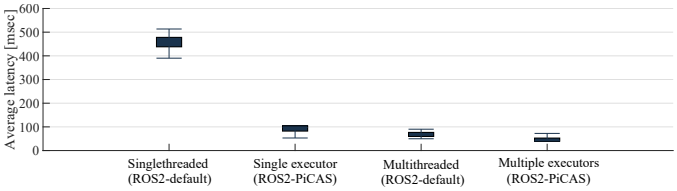

Fig. 3: Average end-to-end latency of hot topic path

d-to-end latency of hot topic path. Fig. 3 illustrates observed average latency of the hot topic path under four different cases. ROS2-PiCAS with a single executor reduces average latency by up to 86% compared to the single-threaded ROS2-default, and shows comparable performance to

the multi-threaded ROS2-default. This result demonstrates the significant benefit of the priority-driven scheduling approach, which help autonomous vehicles recognize obstacles much faster and avoid them in a timely manner while using the same amount of resources.

In case of the multi-threaded executor, ROS2-default performs not as good as ROS2-PiCAS with multiple executors. This is interesting since the default multi-threaded executor follows the global scheduling approach that is naturally better in reclaiming unused resources than partitioned scheduling, which the multiple single-threaded executors of ROS2-PiCAS represent. We suspect that this is not due to an inherent flaw of the multi-thread executor but due to the lack of proper prioritization support.

TABLE I: Number of dropped messages

	Singlethreaded (ROS2-default)	Single executor (ROS2-PiCAS)	Multithreaded (ROS2-default)	Multi. executors (ROS2-PiCAS)
Mean	0.8681	0.0282	0	0
STD	0.3347	0.1651	0	0

Number of dropped messages. Table I shows the number of dropped messages. As expected, ROS2-PiCAS outperforms ROS2-default in a single-threaded executor setup. Note that we do not see any message drops for the multi-threaded ROS2-default and the ROS2-PiCAS with multiple executors.

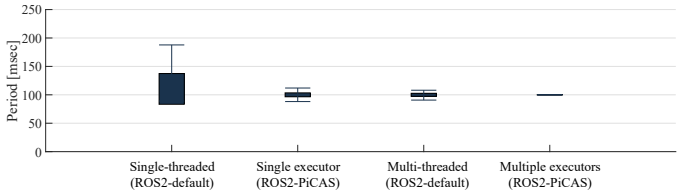

Fig. 4: Behavior Planner jitter

Behavior Planner jitter. Fig 4 illustrates the observed execution period of the Behavior Planner. Any deviation from 100 msec indicates a timing jitter, so a narrow range of observed values is better. As can be seen, ROS2-PiCAS outperforms ROS2-default in all configurations. Such a small uncertainty of the priority-driven scheduling approach can help improve the predictability of the ROS 2 framework.

III. Real-Time Support for Multi-Threaded Executors

Although ROS 2 provides multi-threaded executors, prior studies [5, 6, 8] have considered only single-threaded executors. In general, multithreading allows effective utilization of multiple processors and helps improve system concurrency and throughput. The benefit of real-time multithreading has been demonstrated in the context of self-driving cars [10]. We also have observed that the default multi-threaded executor of ROS 2 has better latency performance than its single-threaded counterpart, as shown in Fig. 3. Despite such a benefit of the ROS 2 multi-threaded executor, to the best of our knowledge, there is no prior work on analyzing and improving the timing behavior of the multi-threaded executor for ROS 2. Therefore, in this section, we discuss challenges that arise with the ROS 2 multi-threaded executor.

In order to make use of the multi-threaded executor in a system with stringent timing requirements, the very first step required is to formally analyze its timing behavior as people did for the single-threaded executor. However, unlike the single-threaded executor, the analysis of processing chains on a multi-threaded executor is more challenging due to the runtime callback distribution across multiple threads and the unsynchronized polling points of the threads. Such challenges makes it difficult to extend the existing ROS 2 analysis techniques to multi-threaded executors directly. For analysis purposes, we are currently modeling single-threaded and multi-threaded executors as partitioned and global schedulers, respectively. Throughout this modeling, we aim to extend the conventional non-preemptive global task scheduling techniques, e.g., [11], to the ROS 2 environment by taking into account semantic differences such as callback dependencies, chains, polling points, and ready set management. We are also working on extending PiCAS to multi-threaded executors to enable priority-driven scheduling and to achieve better end-to-end latency and predictability. Once done, we can compare the performance of priority-driven callback scheduling in a multi-threaded executor vs. in multiple single-threaded executors.

ROS 2 provides an interesting feature for multi-threaded executors, called the *callback group*, which can be used to enforce concurrency rules for callbacks. There are two types of callback groups: *mutually-exclusive* and *reentrant*. Based on the type of the callback groups, the timing behavior of the system and the end-to-end latency of chains will be different. This opens new problems that motivated us to further explore: i) how these callback groups might affect the timing behavior of ROS 2 executors, ii) how we can analytically model the end-to-end latency of chains for each type, and iii) how these can be configured to improve real-time performance. We believe studies on these issues can lead to more efficient scheduling approaches in ROS 2, e.g., assigning callbacks to groups and then scheduling the callback groups.

IV. Challenges with Real-Time GPU Acceleration

This section addresses issues with applications that rely on GPU accelerated kernels. Many applications designed with ROS2 utilize asynchronous and unstructured models for kernel execution on GPU accelerators. While this encourages direct resource allocation and accelerator kernel calling from individual ROS2 nodes, this may incur unpredictable real-time behavior, especially when many nodes request the same accelerator resource. Utilizing shared accelerator resources for complex software stacks, including autonomous vehicle (AV) stacks, is inevitable with modern computer and accelerator architecture. Our on-going work focuses on providing real-time GPU kernel execution management on resource-constrained systems.

A. Problems with Shared Accelerators

With an increasing amount of shared accelerator utilization among complex software stacks, comes consequences that

compromise real-time guarantees for safety-critical workflows. For ROS and ROS2 specific AV stacks, many individual processing chains may necessitate the use of GPU-based accelerators for various perception, localization, mapping, and other tasks. For systems that maintain ample accelerator resources, blocking time for high-priority chains induced by GPU kernel execution from low-priority chains may be uncommon. However, for resource-constrained systems, high-priority chains may suffer from severe delays and deadline misses due to priority inversion when low-priority chains have been already utilizing the shared accelerator resource.

B. Maintaining Real-time Support with Accelerators

A solution that we are currently exploring to address to these dependability issues relies on a GPU-server-based approach within the ROS2 software stack. In conventional autonomous vehicle software design, the callbacks of each node directly invoke the GPU to execute kernels. Our current approach will utilize a separate ROS2 node that acts as a GPU server – handling GPU access requests from all nodes in the stack. This idea is motivated by our earlier work on real-time GPU server [9]. The GPU server architecture will employ priority-based scheduling with support for request-level preemption. We are also considering concurrent kernel execution with real-time spatial GPU multitasking [13, 14] and prioritized CUDA streams [15] for better resource utilization and lower response time. In Fig. 5, describing the overall architecture, ROS nodes will request that a specific GPU kernel be executed on a specific set of data. Intuitively, this will cause additional delays due to extra memory copies between nodes. However, minimizing data copy delays with efficient zero-copy IPC methods like Iceoryx [1] and shared memory transport allows this architecture to support a very low-overhead accelerator resource management framework. The GPU-server node will maintain a structure of all GPU kernels and will schedule the execution of a kernel on a node's data in accordance with the corresponding chain's priority. Handling GPU kernel scheduling in the software stack rather than leaving it to the OS or GPU driver will give applications granular control over how specific chains access the GPU. Other methods of GPU multitasking and scheduling, such as Nvidia's Multi-Process Service (MPS) [12], can allow for multiple processes to perform concurrent kernel execution on different SM's, but do not provide any real-time, priority-based, or preemptive support for processing chains in ROS 2.

V. Conclusion

In this paper, we presented the benefit of enabling priority-driven scheduling in the ROS 2 framework and discussed open challenges. We integrated our prior work on priority-driven chain-aware scheduling into the reference autonomous system and evaluated several key performance indicators under a real-world scenario. The results of the case study demonstrate that the priority-driven scheduling approach significantly outperforms the existing ROS 2 scheduling scheme with respect to the average end-to-end latency, dropped messages, and jitter of

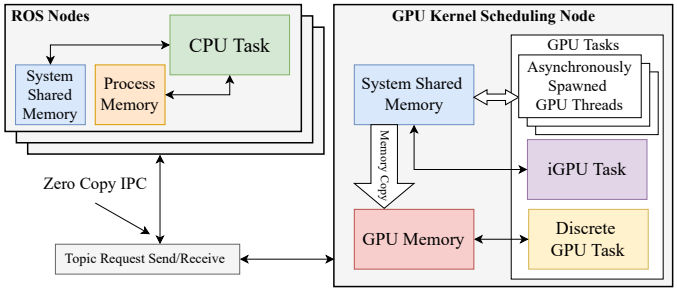

Fig. 5: ROS 2 GPU server framework

periodic nodes. However, previous work, including our own, has been conducted under the assumption of a single-threaded executor, and the extension of existing techniques to the multi-threaded executor still remains as open problems. Besides, real-time support of ROS 2 with shared accelerators such as GPU and FPGA is another challenge that should be resolved for modern intelligent applications. We discussed these challenges and outlined directions to address them following the priority-driven approach.

Our focus in this paper has been around the default ROS 2 executor design and implementation, but there are existing executors, such as the cbg executor [16] and those proposed in the ROS 2 Executor Workshop [2]. We plan to evaluate the effectiveness of our approach against them in the future.

Acknowledgment

We gratefully acknowledge support from the ONR grant N00014-19-1-2496 and the NSF awards 1943265 & 1955650.

References

[1] Eclipse iceoryx - true zero-copy inter-process-communication. https://github.com/eclipse-iceoryx/iceoryx, accessed March 2022.
[2] ROS2 Executor: How to make it efficient, real-time and deterministic? https://www.apex.ai/roscon-21, accessed March 2022.
[3] ROS2 Real-Time Working Group: Reference system. https://github.com/ros-realtime/reference-system, accessed March 2022.
[4] Autoware Foundation. https://gitlab.com/autowarefoundation/autoware.auto, accessed May 2022.
[5] T. Blaß et al. A ROS 2 response-time analysis exploiting starvation freedom and execution-time variance. In *RTSS*, 2021.
[6] D. Casini et al. Response-time analysis of ROS 2 processing chains under reservation-based scheduling. In *ECRTS*, 2019.
[7] H. Choi et al. Chain-based fixed-priority scheduling of loosely-dependent tasks. In *ICCD*, 2020.
[8] H. Choi et al. PiCAS: New design of priority-driven chain-aware scheduling for ROS2. In *RTAS*, 2021.
[9] H. Kim et al. A server-based approach for predictable GPU access with improved analysis. *Journal of Systems Architecture*, 88:97–109, 2018.
[10] J. Kim et al. Parallel scheduling for cyber-physical systems: Analysis and case study on a self-driving car. In *ICCPS*, 2013.
[11] J. Lee. Improved schedulability analysis using carry-in limitation for non-preemptive fixed-priority multiprocessor scheduling. *IEEE Transactions on Computers*, 66(10):1816–1823, 2017.
[12] Nvidia. Nvidia multi-process service. https://docs.nvidia.com/deploy/mps/index.html, accessed March 2022.
[13] S. Saha et al. STGM: Spatio-temporal GPU management for real-time tasks. In *RTCSA*, 2019.
[14] Y. Wang et al. Balancing energy efficiency and real-time performance in GPU scheduling. In *RTSS*, 2021.
[15] Y. Xiang and H. Kim. Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference. In *RTSS*, 2019.
[16] Y. Yang and T. Azumi. Exploring real-time executor on ROS 2. In *ICESS*, 2020.

Minimal-Overlap Centrality-Driven Gateway Designation for Real-Time TSCH Networks

Miguel Gutiérrez Gaitán*†‡
*Facultad de Ingeniería
Universidad Andres Bello
Santiago, Chile
miguel.gutierrez@unab.cl

Pedro M. d'Orey
†CISTER Research Center
Universidade do Porto
Porto, Portugal
ore@isep.ipp.pt

Pedro M. Santos
†CISTER/ISEP
Politécnico do Porto
Porto, Portugal
pss@isep.ipp.pt

Luís Almeida
‡CISTER/FEUP
Universidade do Porto
Porto, Portugal
lda@fe.up.pt

***Abstract*—This research proposes a novel *minimal-overlap* centrality-driven gateway designation method for real-time wireless sensor networks (WSNs). The goal is to enhance network schedulability by design, particularly, by exploiting the relationship between path node-overlaps and gateway designation. To this aim, we define a new metric termed *minimal-overlap network centrality* which characterizes the *overall overlapping* degree between all the active flows in the network when a given node is selected as gateway. The metric is then used to designate as gateway the node which produces the least overall number of path overlaps. For the purposes of evaluation, we assume a time-synchronized channel-hopping (TSCH) WSN under centralized earliest-deadline-first (EDF) scheduling and shortest-path routing. The assessment of the WSN traffic schedulability suggests our approach is dominant over classical network centrality metrics, namely, eigenvector, closeness, betweenness, and degree. Notably, it achieves up to 50% better schedulability than a degree centrality benchmark.**

***Index Terms*—Centrality, Network design, TSCH, WSN.**

I. Introduction

The Industrial Internet of Things (IIoT) is dramatically increasing the adoption of wireless technologies in several industries [1]. Wireless sensor networks (WSNs), as one of the key enabling technologies for IIoT, allows gathering (wirelessly) critical sensor data in a variety of industrial fields [2], ranging from manufacturing to automotive. Time-synchronized channel-hopping (TSCH) is one of the major multi-channel medium access control (MAC) protocols for industrial WSNs offering improved reliability and support for real-time communication [2]–[4].

Industrial WSNs are usually formed by tens to hundreds of devices that deliver *deadline-constrained* sensor data toward a common gateway [3]. The gateway - an essential node enabling seamless communication with external entities - also plays a role in real-time network operation. In particular, our recent study on TSCH WSNs [5] has shown that a simple but rather effective criterion for gateway designation can remarkably enhance real-time WSN performance by design. Resorting to the notion of *network centrality* (i.e., a relative measure of the importance of the node according to its position in the network), the authors explored common metrics from social network analysis for improved schedulability. Despite the promising results, none of the assessed metrics dominated over the others and optimal performance was far from being achieved. A challenge we attempt to address herein.

We propose a novel *centrality-driven gateway designation* method for real-time WSNs based on the reduction of path node-overlaps in shortest path routing. We deal with alike foundational questions of work [5], but we solve the *gateway designation* problem by proposing a novel flow-informed metric termed *minimal-overlap* centrality. This metric requires knowing the routing approach beforehand to compute the overall *overlapping degree* resulting from the encountering of all active flows in the network elements. This measure is inspired by the *minimal-overlap* routing protocol [4], which reduces the overall overlapping degree of the network using a greedy heuristic that weights the network links based on the node-overlaps between flows.

By contrast, this work reduces the network global overlapping degree by judiciously choosing as gateway the node that *minimizes* the overall number of overlaps. While a schedulability-optimal choice could be made using enough computational power, we explore here a less demanding method that does not require fully assessing network schedulability to achieve near optimal real-time performance. The method resorts to shortest path routing for simplicity, but the concept can be easily extended to different routing schemes without loss of generality. To our knowledge, this is the first centrality-driven gateway designation method specifically designed to reduce end-to-end deadline misses in WSNs.

II. System Model

A. Wireless Network

The communication network is abstracted as an undirected graph $G = (V, E)$ where V is the set of vertices or nodes and E is the set of edges or links between those vertices. The order

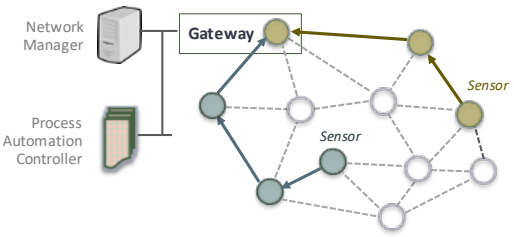

Fig. 1: An illustration of a multi-hop WSN.

of graph G is denoted as $N = |V|$, of which a set of $N-1$ nodes act as sensor nodes while one node acts as a gateway. The gateway node is used for enabling communication with external entities (e.g., network manager) forming a wireless mesh network with the remaining nodes (Fig. 1).

Multiple access is governed using TSCH protocol, one of the operating modes of IEEE 802.15.4 standard. TSCH uses fixed size TDMA slots combined with multi-channel hopping, allowing concurrent transmission over up to $m = 16$ different channels with global synchronization. A time slot allows transmitting a single packet and receiving the corresponding acknowledgement. All packet transmissions are managed centrally using an earliest-deadline-first (EDF) scheduling policy and a (hop-count) shortest path routing algorithm.

B. Real-Time Flows

In terms of traffic flow, a subset of sensor nodes transmit (potentially an infinite number of) *deadline-constrained* data with a fixed period T_i; the remaining nodes act as relays to transmit it towards the gateway. The resulting set of n real-time data flows is denoted as $F = \{f_1, f_2, \ldots, f_n\}$. Each data flow is characterized by a 4-parameter tuple $f_i = (C_i, D_i, T_i, \phi_i)$, where C_i is the transmission time between the source node s_i and the gateway, D_i is the (relative) deadline, and ϕ_i is the multi-hop routing path. The term $f_{i,\gamma}$ represents the γ^{th} transmission of flow f_i released at time $r_{i,\gamma}$ such that $T_i = r_{i,\gamma+1} - r_{i,\gamma}$. $f_{i,\gamma}$ is constrained to reach the gateway before its absolute deadline $[d_{i,\gamma} = r_{i,\gamma} + D_i]$.

C. Real-Time Performance

The real-time performance of the centralized TSCH network under EDF [6] is evaluated resorting to the supply/demand-bound based schedulability analysis presented in [7]. The method evaluates if the *supply-bound function* (sbf) [i.e., minimal transmission capacity offered by a WSN with m channels] is equal or larger than the *forced-forward demand-bound function* [8] for WSN (FF-DBF-WSN) [i.e., upper bound on the maximum demand imposed by a set of n time-sensitive flows assessed in any interval of length ℓ]. Formally, this WSN traffic schedulability test is posed as:

$$\text{FF-DBF-WSN}(\ell) \leq \text{sbf}(\ell),\ \forall \ell \geq 0 \tag{1}$$

The $\text{sbf}(\ell)$ is such that satisfies the following conditions:

$$\text{sbf}(0) = 0 \wedge \text{sbf}(\ell + h) - \text{sbf}(\ell) \leq m \times h, \forall \ell, h \geq 0 \tag{2}$$

The upper bound on network demand FF-DBF-WSN [7] is composed by two terms, namely i) **channel contention** (i.e., accounts for mutually exclusive scheduling on multiple channels, being equivalent to FF-DBF for multiprocessors [8]) and ii) **transmission conflicts** (i.e., delay contribution due to multiple flows encountering on a common half-duplex node):

$$\text{FF-DBF-WSN}(\ell) = \overbrace{\frac{1}{m}\sum_{i=1}^{n} \text{FF-DBF}(f_i, \ell)}^{\text{CHANNEL CONTENTION}} + \underbrace{\sum_{i,j=1}^{n} \left(\Delta_{i,j} \cdot \max\left\{ \left\lceil \frac{\ell}{T_i} \right\rceil, \left\lceil \frac{\ell}{T_j} \right\rceil \right\} \right)}_{\text{TRANSMISSION CONFLICTS}} \tag{3}$$

where $\Delta_{i,j}$ is a path overlapping factor between any pair of flows f_i and $f_j \in F$ (with $i \neq j$) as defined in [9]. Formally, this factor is defined as:

$$\Delta_{i,j} = \sum_{k=1}^{\delta(ij)} Len_k(ij) - \sum_{k'=1}^{\delta'(ij)} (Len_{k'}(ij) - 3) \tag{4}$$

where $\delta(ij)$ indicates the total number of overlaps between f_i and f_j of which $\delta'(ij)$ are the ones larger than 3. The length of the k^{th} and k'^{th} path overlap between f_i and f_j are named $Len_k(ij)$ and $Len_{k'}(ij)$, respectively, with $k \in [1, \delta(ij)]$ and $k' \in [1, \delta'(ij)]$. In the convergecast case all paths are directed to the root and thus only one path of arbitrary length is shared between any pair of flows.

III. Minimal-Overlap Centrality-Driven Gateway Designation for Real-Time WSNs

Given the system model presented in Section II, we consider the problem of how to designate a node as gateway for improved WSN traffic schedulability. To this purpose, resorting to the notion of *network centrality*, we propose a new centrality metric that characterizes the relationship between the *overall* path node-overlaps and gateway designation. Similarly to [5], the proposed metric is then used to designate as gateway the node with the highest centrality score. Classical network centrality metrics are also considered for benchmarking purposes.

A. Minimal-Overlap Network Centrality

Specifically, we propose a new network centrality metric termed *minimal-overlap* (MO) centrality. This metric is built upon the computation of the overall *path overlapping* resulting from the superposition of all flow routes in the network when directed to a given node $v_q \in V$. The importance (centrality) of the node v_q is reflected by the following expression:

$$MO(v_q) = \frac{1}{\sum_{i,j=1 \wedge i \neq j}^{n} \Delta_{i,j}^{q} + 1} \tag{5}$$

where the factor $\Delta_{i,j}^{q}$ is the overlap contribution from flows f_i and f_j (Eq. 4) when their routes ϕ_i and ϕ_j are directed toward node v_q, and n is the number of flows in the set F. Note that we consider only a subset of $N-n$ nodes as gateway candidates v_q, since the remaining are defined as sources. Without loss of generality, we also assume the routes are computed using a hop-count-based *shortest-path* algorithm.

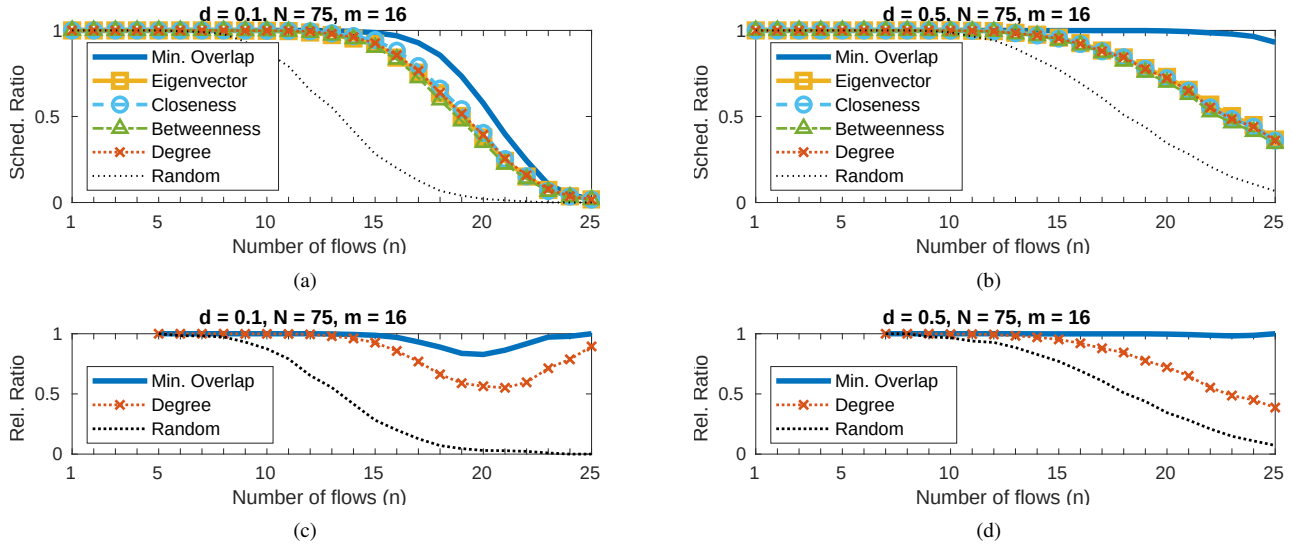

Fig. 2: **Top**: Schedulability ratio of 1000 random topologies for different number of flows with target density 0.1 (a) and 0.5 (b) resorting to the gateway designation methods based on i) classical network centrality metrics (e.g. degree centrality) and ii) the proposed minimal-overlap network centrality. **Bottom:** the deviation in terms of schedulability from the best and worst possible gateway assignments.

B. Classical Network Centrality Metrics

For comparison, we consider the 4 most common network centrality metrics in the literature, namely, eigenvector, closeness, betweenness and degree. For completeness, the definitions of those metrics are given in Table I [1]. More details on these metrics within the context of gateway designation in real-time WSNs can be found in [5].

TABLE I: Classical Centrality Metrics.

Metric	**Definition**
Eigenvector	$EC(v_q) = \frac{1}{\lambda_{max}(A)} \cdot \sum_{j=1}^{N} a_{j,q} \cdot x_j$
Closeness	$CC(v_q) = \frac{1}{\sum_{p \neq q} distance(v_p, v_q)}$
Betweenness	$BC(v_q) = \sum_{q \neq r} \frac{sp_{r,s}(v_q)}{sp_{r,s}}$
Degree	$DC(v_q) = \frac{degree(v_q)}{N-1}$

IV. Performance Evaluation

A. Simulation Setup

Wireless network. We consider 1000 random topologies built using a synthetic generator of network graphs. Each topology is generated with a target node density d using a sparse uniformly distributed random matrix with dimension $N \times N$. We use $N = 75$ for all experiments. We consider that the TSCH network operates with $m = 16$ channels.

Network flows. A subset of $n \in [1, 25]$ vertices of G is chosen randomly as source nodes transmitting periodically deadline-constrained data to the gateway. The parameters of each data flow $f_i = (C_i, D_i, T_i, \phi_i)$ are defined as follows. C_i is computed by multiplying the time slot duration (10 ms) with the number of hops in the path ϕ_i. D_i is set in implicit-deadline model, i.e. $D_i = Ti$. T_i is harmonic and randomly generated in the range of $[2^4, 2^7]$ as in [5]. This implies a super-frame length of $H = 1280$ ms.

Real-time assessment. We assess schedulability over a time interval equal to the super-frame, i.e., $\ell = H$, and when all the $m = 16$ channels are available. EDF and shortest path routing are assumed for all transmissions. Concerning $\Delta_{i,j}$, we use precise computation derived from the network topologies.

B. Results & Discussion

Schedulability Analysis. Fig. 2 presents the schedulability ratio as a function of the number of flows considering two network densities, namely 0.1 (left) and 0.5 (right), and different methods for gateway designation, namely minimal-overlaps and classic network centrality-based. We also compute the best and worst schedulability-driven gateway selections obtained with extensive search as well as a random selection. As expected, the schedulability ratio decreases for larger number of flows in all configurations due to the larger channel contention and transmission conflicts. Conversely, higher network density increases the number of potential paths between any given pair of nodes, favoring schedulability.

The results show that the minimal-overlap gateway designation method achieves higher schedulability for all numbers of flows and densities when comparing with a method based on classical centrality metrics (e.g. degree or betweenness centrality). We argue this is caused by the MO method decreasing, by design, the number of overlapping paths allowing to reduce transmission conflicts (Fig. 3), thus improving the timely delivery of data. As expected, the proposed method is also

[1]**Notation.** EC: $\lambda_{max}(A)$ is the largest eigenvalue of the adjacency matrix $A = [a_{j,q}]_N$, where $a_{j,q}$ is the matrix element at row j and column q, and x_j is the jth value of the eigenvector x of graph G. CC: $distance(v_p, v_q)$ is the shortest-path (hop-count) distance between vertices v_p and v_q, with $p \neq q$, $\forall\ v_p \in V$. BC: $sp_{r,s}$ is the number of shortest paths between any pair of vertices v_r and v_s, and $sp_{r,s}(v_q)$ is the number of those paths passing through node v_q; DC: $degree(v_q)$ denotes the number of edges of node v_q which are directly connected to any of the rest $N-1$ nodes in the graph G.

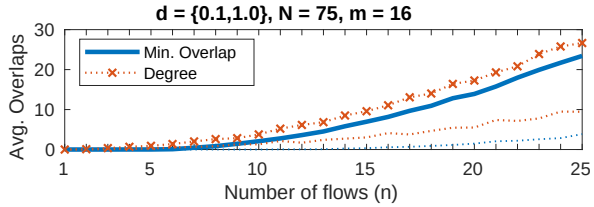

Fig. 3: Average number of overlaps of 100 random topologies when varying network flows for two gateway designation methods and two extreme densities, namely 0.1 (solid line) and 1.0 (dotted line).

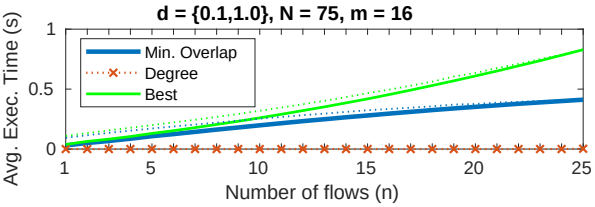

Fig. 4: Execution time for different gateway designation methods considering up to 25 network flows and two extreme target densities, namely 0.1 (solid line) and 1.0 (dotted line).

clearly superior to the random baseline, further demonstrating the significance of judicious gateway designation.

We also analyze how the proposed method deviates from the system optimal gateway election (Figs. 2c and 2d). The metric *relative ratio* is defined as the ratio between the schedulability ratio of a given method to the schedulability ratio of the best and worst performing nodes in the network, with a value of 1 denoting best and 0 the worst performance. The results show the performance of the proposed method is only slightly below the best method, having the maximum degradation of $\sim$20% for a density of 0.1 and 20 simultaneous flows. We highlight this degradation is negligible for larger densities (e.g. $d = 0.5$) since the overall overlapping degree decreases for increasing density (Fig. 3), which was also confirmed by previous studies [4]. Finally, the results in Fig. 2 also reveal the performance improvements of the proposed method, in general, increase for higher density and higher number of flows when comparing with other centrality-based metrics or random gateway assignment.

Computational Cost. Fig. 4 depicts the average execution time for the different gateway designation methods and the optimal gateway designation. Regarding the classical centrality-based designation method, we solely present the result for degree centrality for visual clarity and because this has the lowest execution time among all metrics. We also present result for two extreme density values of 0.1 (solid line) and 1 (dashed lines). The setup for this experiment used MATLAB R2020b on Ubuntu 18.04 LTS on a laptop with an Intel Core i7-6500U CPU at 2.5GHz and 4GB of DD3 RAM.

The results confirm the low execution time of the degree-centrality gateway designation. On the other hand, minimal overlaps designation considerably decreases the execution time when compared against optimal gateway designation, particularly for higher number of flows. Note that the optimal method uses extensive search with full schedulability analysis for each case, while the MO metric just requires computing the number of overlaps in the network given a set of flows. The results also show that the density has a minimal impact on the average execution time. Overall, the proposed method provides a good trade-off between achievable schedulability ratio (near optimal) and computational cost (about half the value of the optimal method).

V. Conclusions

This paper presented a novel gateway designation method for real-time WSNs based on minimizing the number of path overlaps in the network. Simulation results show improved schedulability ratio when comparing with other classic centrality-based gateway selection methods, achieving nearly optimal performance under specific conditions (e.g. larger network densities) while showing lower execution times than the optimal case. As future work, we intend to extend the method to multiple gateways, as well as to evaluate its applicability in the context of wireless edge-node placement.

Acknowledgment

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UIDB/04234/2020); by the Operational Competitiveness Programme and Internationalization (COMPETE 2020) under the PT2020 Agreement, through the European Regional Development Fund (ERDF); also by FCT and the ESF (European Social Fund) through the Regional Operational Programme (ROP) Norte 2020, under PhD grant 2020.06685.BD.

References

[1] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE Trans. on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[2] D. Dujovne, T. Watteyne, X. Vilajosana, and P. Thubert, "6TiSCH: deterministic IP-enabled industrial internet (of things)," *IEEE Communications Magazine*, vol. 52, no. 12, pp. 36–41, 2014.

[3] J. Wang, T. Zhang, D. Shen, X. S. Hu, and S. Han, "APaS: An adaptive partition-based scheduling framework for 6TiSCH networks," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 320–332, IEEE, 2021.

[4] M. G. Gaitan, L. Almeida, P. M. Santos, and P. M. Yomsi, "EDF scheduling and minimal-overlap shortest-path routing for real-time TSCH networks," in *Workshop on Next Generation Real-Time Embedded Systems (NG-RES)*, vol. 87, (Virtual), pp. 2–1, 2021.

[5] M. G. Gaitan, L. Almeida, A. Figueroa, and D. Dujovne, "Impact of network centrality on the gateway designation of real-time TSCH networks," in *IEEE Int. Conference on Factory Communication Systems (WFCS)*, pp. 139–142, 2021.

[6] M. G. Gaitan, P. M. Yomsi, P. M. Santos, and L. Almeida, "Work-in-progress: Assessing supply/demand-bound based schedulability tests for wireless sensor-actuator networks," in *IEEE Int. Conference on Factory Communication Systems (WFCS)*, pp. 1–4, IEEE, 2020.

[7] M. G. Gaitan and P. M. Yomsi, "FF-DBF-WIN: On the forced-forward demand-bound function analysis for wireless industrial networks," in *Work-in-Progress Session of the 30th Euromicro Conference on Real-Time System (ECRTS)*, pp. 13–15, 2018.

[8] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller, "Improved multiprocessor global schedulability analysis," *Real-Time Systems*, vol. 46, no. 1, pp. 3–24, 2010.

[9] C. Xia, X. Jin, and P. Zeng, "Resource analysis for wireless industrial networks," in *Int. Conference on Mobile Ad-Hoc and Sensor Networks (MSN)*, pp. 424–428, IEEE, 2016.

No-more-unbounded-blocking queues: bounding transmission latencies in real-time edge computing

Gabriele Serra
Scuola Superiore Sant'Anna
Pisa, Italy
gabriele.serra@santannapisa.it

Pietro Fara
Scuola Superiore Sant'Anna
Pisa, Italy
pietro.fara@santannapisa.it

***Abstract*—Nowadays, with advancements in computing power and energy efficiency, embedded system platforms are becoming able to provide services that, previously, required the cloud to be served. Accordingly, the edge computing paradigm is becoming increasingly popular, as it allows, among other advantages, to foster security and privacy preservation by processing data at the origin. On the other hand, these systems demand predictability across the edge-to-cloud continuation. Regardless of the communication link used by the edge node, tasks send out data employing a transmission queue; for system-designer, analyzing the time behavior of a task becomes challenging when each task has to wait a variable amount of time to send a packet. This work presents a model to analyze the different sources of latency introduced when dealing with a communication interface. The mentioned model ensures that the data traffic does not exceed the transmission queue limit to avoid unbounded blocking on task execution.**

I. Introduction

Embedded systems are every day more pervasive in human-beings lives: they are used in any kind of environment, ranging from agriculture to transports; they fulfill evermore functions, of which some are related to the safety of environments or people. Further, with technology advancements, embedded system platforms are becoming more powerful and they are able to provide services that, until few years ago, required server to be accomplished. Therefore, the edge computing paradigm is becoming increasingly popular. Processing data closer to its origin drastically reduces the amount of data sent to the cloud and, therefore, facilitates real-time computation, reduces energy consumption (together with carbon footprint) and help preserving data privacy. However, when the functions performed by the edge-computing system need to interact with the external environment, the predictability across the edge-to-cloud continuum is a key requirement. In the meantime, the use of system resources must be kept efficient.

Especially when dealing with connected systems, the edge computing nodes are required to deliver a result within a predefined deadline; the result typically consist in a chunk of data to be transmitted across the network. Regardless the communication mechanism used by the node, commonly the sender task places the packet in a transmission queue. When the queue reaches the maximum queue size, the task must wait an unbounded amount of time to transmit the packet. Furthermore, the device transmission protocol takes some time to be performed. Indeed, as a consequence, the communication device introduces latencies between the edge node and the cloud. This makes particularly challenging to analyze the time behavior of the system. While it is not possible to bound the delay introduced by a complex network such as the global internet, it is interesting to analyze the type of latencies introduced by the transmission device and how these latencies must be accounted when analyzing the timing behavior of the edge node.

Contribution. In summary, this work makes the following contributions:

- It presents a detailed analysis of the latencies that a packet can experience during the transmission phase in an edge node
- It derives an analysis that prevent device transmission queue to be full, in order to bound the time required for a task to send data through the communication peripheral

Paper structure. The remainder of this paper is organized as follows. Section II reviews the related work. Section III presents the system by considering task model and communication assumptions. Section IV analyzes queueing effects and delays in inter-replica communications and Section V concludes the paper.

II. Related Work

In the last few years, edge computing has become more popular due to the increase in performance and the decrease in the size of microcontrollers and devices. A particular architecture, called Mobile Edge Computing (MEC), is gaining more attention from researchers, especially for the massive diffusion of Internet-of-things (IoT) devices. An overview of this new architecture [8] was presented in 2016. Based on MEC, many studies have been done to improve the overall performance of edge-to-cloud applications in terms of latency. Ren et al. [7] formulated an optimization problem to find the better solution for splitting the entire computation of a single task into two different parts: the first one to be executed at the edge and the second one on the cloud. Another work on latency optimization based on data compression was presented in 2018 [6]: they analyzed three different computation models: local compression, edge-cloud compression and partial compression offloading. In the last model, they formulated an optimization problem to find the optimal solution in terms of latency. Latency may also increase because a lot of computation is

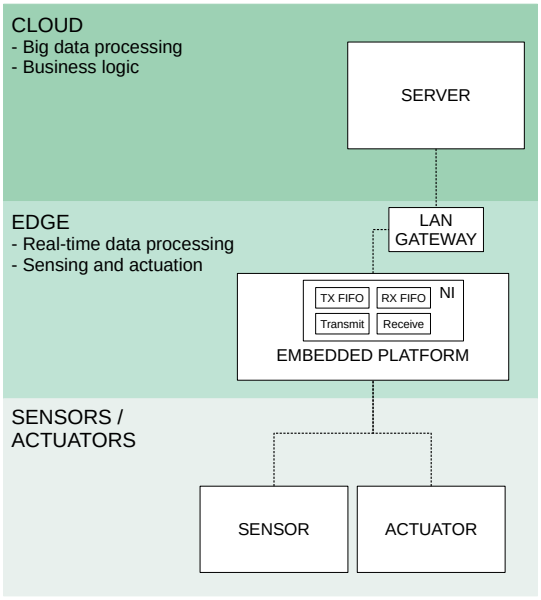

Fig. 1. An overview of the system architecture.

needed to provide services on mobile applications. In [4], Jia et al. proposed a heuristic offloading method for computation-intensive applications in MEC, taking into account the cloud service latencies and computation time and improving it with load-balancing techniques. In 2017, LAVEA platform [10] was designed to offload computation between clients and edge nodes and to let nearby edge nodes collaborate to provide low-latency video analytics at places closer to the users. Latency is also depending on how transmission queues are managed. In [1], Aamir et al. proposed MANET: a new scheme to improve packet queues management in terms of packet loss ratio. Another queue management mechanism, based on the division of the main queue into two sub-queues, has been developed in [9].

III. System model

This work focuses on analyzing delays introduced during data transmission from an edge node. As multi-core interference is outside the scope of this work, the edge node consists of an embedded system platform with a single processor. A set of n periodic tasks $\{\tau_1, \ldots, \tau_n\}$ is executed on the mentioned processor. Tasks are scheduled through a fixed-priority preemptive scheduling algorithm and activated all at the same time without any initial offset. Hence, each periodic task τ_i, is characterized by a worst-case execution time C_i, a release period T_i, and a relative deadline $D_i \leq T_i$. We denote with H the hyper-period among all periods of tasks τ_i.

The node is connected to a communication network `CN`. To be as general as possible, this work does not take into account a specific network link. The edge node exposes, on the `CN`, an interface capable of sending data and receiving it. From now on, we will denote the exposed interfaces of the node as the node-interface `NI`. Data transmission via the `CN` occurs by acting on a few memory-mapped device registers. An overview of the architecture of the system considered by our model is shown in Figure 1.

The `NI` provides an output (and respectively, an input) buffer organized as a first-in-first-out (FIFO) queue of q^{NI} elements, each sized b bytes. Accessing such registers consists of performing several writing and reading operations on the memory-mapped device registers. The minimum read/write rate to access such registers is indicated with β (bytes per time unit), while the maximum rate is denoted by β^{max}. Furthermore, the `NI` guarantees a minimum transmission rate on the link that, in this model, is indicated by α (bytes per time unit).

To send data out from `NI`, the content of a packet must be copied from the task memory to the device queue. Memory write times are generally shorter than read times; however, we denote with γ minimum read/write rate to access memory. The model considers a matching rate for read/write operations as it does not particularly affect the results of this work.

When the `NI` transmission queue is full, if a task wants to send data must wait until one of the slots in the queue becomes empty. For the sake of our model, we are not interested in analyzing how the `NI` operates when receiving data.

To summarize, when a task has to send x bytes out, the `NI` needs **(i)** at most x/γ time units to read the data from the task memory, **(ii)** at least x/β^{max} time units and at most x/β time units to copy the mentioned data into the `NI` queue and **(iii)** at most x/α time units to transmit such data into the communication link.

A task τ_i exchanges M_i data packets with a fixed size of b bytes. Note that not all the periodic tasks may produce data that must be sent over the network; hence M_i can also be null.

Data transmission is performed through the `NI` in a mutual-exclusive way. Thus, each task may have to acquire and release a lock before and after transmitting each packet. Our model adopts the immediate priority ceiling (IPC) locking protocol to avoid priority inversion phenomena. In the case in which all M_i are different from zero, `NI` is equivalent to a resource shared by all tasks under the IPC protocol. Therefore, the critical sections due to the `NI` access are practically non-preemptive.

We indicate the average amount of bytes transmitted through the `NI` as $\overline{B}(t)$.

IV. Transmission analysis

This section will analyze the type of latencies introduced by the `NI`, deriving a condition to ensure that the number of packets sent through the `NI` peripheral does not exceed the queue limit to avoid introducing unbounded blocking.

A. Latency modeling

Definition 1: Δ_{NI} indicates the amount of time that elapses since a packet is stored in the transmission queue by the sender task to the time the packet can be considered sent out and

thus available for the subsequent node of the network (that, commonly, is the network edge router).

In the following subsection, the latency introduced by NI is studied employing a model commonly used to account for delays in routers network [2] [5]. Under this model, Δ_{NI} can be decomposed as a sum of four different components: propagation latency d_{prop}, transmission latency d_{trans}, processing latency d_{proc}, queuing latency d_{queue}.

Therefore, the total latency due to the NI packet transmission can be computed as:

$$\Delta_{NI} = d_{prop} + d_{trans} + d_{proc} + d_{queue}$$

The purpose of this work is not to provide an estimation of the latency Δ_{NI} but to demonstrate that, adopting our methodical analysis, that latency can be bounded. In order to show that for Δ_{NI} exists an upper bound Δ_{NI}^{max}, we have to analyze every single component. In the following subsections, we will investigate all the sources of latency characterizing each one.

1) Propagation latency: Once bits are pushed in the CN link, they need to propagate to the other end. Therefore, the propagation latency depends on the channel length and the signal propagation speed for the given medium. Commonly, propagation latency can be computed as the distance between devices divided by channel propagation speed. The signal propagation speed is affected by multiple factors. Although some specific wireless networks (e.g., wireless acoustic networks) may have a propagation latency up to several milliseconds, in the majority of cases, if the node is connected by means of a network cable or a wireless network, the latency due to the propagation speed lies in the order of nanoseconds. Accordingly, in the following sections, we will neglect the propagation latency.

2) Transmission latency: The transmission latency is an amount that represents the time required by the device to transmit a chunk of bytes into the link and depends on the device bandwidth. It is computed as the number of bytes to be pushed in the link over the transmission bandwidth. Given that each task exchanges data packets with a fixed size of b bytes, we may indicate d_{trans} as: $d_{trans} = \frac{b}{\alpha} = \sigma_d$ with σ_d being a constant factor to account for the transmission time demanded by each packet.

3) Processing latency: The processing latency represents the time required to read and write device registers and performing register shifts. It is fixed for each packet and does not depend on the packet length. Therefore we denote $d_{proc} = \sigma_o$ with σ_o being the maximum data-size-independent processing overhead.

4) Queueing latency: The queueing latency represents the amount of time each packet must wait in the queue to be transmitted in the link. When the queue is empty, and no other packet is being transmitted, the queueing latency will be null. Otherwise, a packet's queueing latency depends on the number of currently enqueued packets that wait for transmission.

Assuming that each packet transmission costs the equivalent of the transmission latency d_{trans} plus the processing latency d_{proc} , the $N-th$ packet in queue will wait $(N-1)\cdot(d_{trans}+d_{proc})$. Therefore, we can compute the maximum queueing latency as: $d_{queue} = (q^{\text{NI}} - 1) \cdot (\sigma_d + \sigma_o)$.

Provided all the mentioned latencies definition, the upper bound Δ_{NI}^{max} can be computed as: $\Delta_{NI}^{max} = q^{\text{NI}} \cdot (\sigma_o + \sigma_d)$

B. Queueing analysis

In our transmission model, we consider the NI queue as a finite queue. Therefore, when the queue is full, a task cannot push another packet into it, remaining blocked on the send operation. A task must perform the send operation on a queue with available free slots to avoid unbounded blocking.

Let's assume that the queue can host an infinite number of packets for the sake of proving the following lemma. Recalling the notation used above, that indicates the transmission bandwidth with α and the average amount of bytes transmitted through the NI as $\overline{B}(t)$, we define the transmission interface utilization average $\overline{U}^{\text{NI}}(t)$ as $\overline{U}^{\text{NI}}(t) = \frac{\overline{B}(t)}{\alpha}$

Lemma 1: The latency component caused by queuing d_{queue} can be bounded during the entire system service only if

$$U^{\text{NI}} = \lim_{t \to H} \overline{U}^{\text{NI}}(t) \leq 1$$

Proof.

Being α a constant, if $\lim_{t\to H} \frac{\overline{B}(t)}{\alpha} > 1$, it means $\lim_{t\to H} \overline{B}(t) > \alpha$, therefore the average rate at which bytes arrives at the queue exceeds the rate at which the bytes can be transmitted by the NI device. Consequently, the queue will tend to grow with no bound, making the queuing latency infinite.

□

In our subsequent timing analysis, we assume as a necessary condition, that U^{NI} must be less or equal than 1. Given that all tasks τ_i are periodic, U^{NI} can be computed as $U^{\text{NI}} = \sum_{i=1}^{n}(M_i b)/T_i$. When the condition mentioned above holds, the nature of $\overline{U}^{\text{NI}}(t)$ impacts the latency. When packets arrive every $\frac{b}{\alpha}$, then each packet finds the queue empty. However, packets can arrive simultaneously, in bursts. Hence, U^{NI} cannot be used to characterize the queuing latency fully. We must be sure that the actual packet burst does not overpass the available queue slots.

We begin by bounding the amount of data sent within arbitrary time windows.

Lemma 2: In any time window of length t, the tasks can provide in the NI queue at most $g(t)$ bytes of data, where

$$g(t) = \min\left\{\sum_{i=1}^{n}\left\lceil\frac{t+T_i}{T_i}\right\rceil M_i b,\ \beta^{max} t\right\}. \qquad (1)$$

Proof.

A periodic task τ_i, in any time window of length t, can release at most $\lceil(t+T_i)/T_i\rceil$ jobs (e.g., see [3]). Each job

of the tasks sends at most M_i packets, each of size b bytes. Hence the first term in the minimum of Eq. (1). Note that the amount of data the tasks can send within a time window is also limited by the maximum rate with which the NI queue can be filled, which is given by β^{max}. The lemma follows. □

The above lemma can then be used to derive a safe condition under which the NI queue is never full.

Lemma 3: No task can find the NI queue full if

$$\forall t > 0, \quad g(t) - \alpha t \leq q^{\text{NI}} \cdot b. \tag{2}$$

Proof.
Assume by contradiction that at a certain time instant t_1 a task finds a NI queue full. Let $t_0 < t_1$ be the latest time at which the NI queue has been empty and let $t = t_1 - t_0$. It holds that $(t_0, t_1]$ is an interval of length t in which the NI has always been busy with packets to transmit. Let $x(t)$ be the amount of bytes issued by the tasks to be provided in the NI queue in $(t_0, t_1]$. Note that during this interval the NI must have sent at least αt bytes: hence, if the queue is full at time t_1 it holds that $x(t) - \alpha t > q^{\text{NI}} \cdot b$.

By Lemma 2, in any time window of length t the cumulative amount of bytes provided in the NI queue is bounded by $g(t)$. Hence, $g(t) \geq x(t)$, which implies $g(t) - \alpha t > q^{\text{NI}} \cdot b$. This contradicts Eq. (2). Hence the lemma follows. □

Note that Lemma 3 does not consist in a practical test as any possible value of t shall be checked. This issue is solved below by limiting the test to a finite number of check-points.

Lemma 4: Lemma 3 holds also if $\forall t \in \Phi, \quad g(t) - \alpha t \leq q^{\text{NI}} \cdot b$, where

$$\Phi = \bigcup_{i=1}^{n} \{kT_i + \epsilon \leq t^*, k = 0, 1, 2, \ldots\} \cup \{\psi\} \tag{3}$$

with

$$t^* = \frac{2\sum_{i=1}^{n} M_i b}{\alpha - \sum_{i=1}^{n} \frac{M_i b}{T_i}} \tag{4}$$

$$\psi = \left\{ t \leq t^* \;\; | \;\; \sum_{i=1}^{n} \left\lceil \frac{t + T_i}{T_i} \right\rceil M_i b = \beta^{max} t \right\} \tag{5}$$

and $\epsilon > 0$ arbitrarily small.

Proof.
We prove the lemma by showing that function $g(t) - \alpha t$ can be maximal only for values $t \in \Phi$. First note that the minimum of two functions is upper bounded by the upper bound of one of the two functions. Let's denote with $G(t) = \sum_{i=1}^{n} \left(\frac{t+T_i}{T_i} + 1 \right) M_i b$, hence $g(t) \leq G(t)$. Denoting $m = \sum_{i=1}^{n} \frac{M_i b}{T_i}$ and $q = 2\sum_{i=1}^{n} M_i b$ we can write $G(t) = mt + q$.

Note that both $G(t)$ and αt are two lines with slope $U^{\text{NI}} = \sum_{i=1}^{n} (M_i b)/T_i$ and α, respectively. Recall that $\alpha > U^{\text{NI}}$ (see Lemma 1). Therefore $G(t)$ and αt intersect and, from their intersection on, we have $g(t) \leq G(t) \leq \alpha t$ and hence also $g(t) - \alpha t \leq 0$. The intersection occurs for the value t^* such that $G(t^*) = \alpha t^*$ and can be computed by solving the latter equality with respect to t^*,

$$mt^\star + q = \alpha t^\star, \quad t^\star = \frac{2\sum_{i=1}^{n} M_j b}{\alpha - \sum_{i=1}^{n} \frac{M_i b}{T_i}}$$

Hence getting the expression at the Eq. (4). Therefore, for values of $t > t^*$ function $g(t) - \alpha t$ cannot be maximal.

If $g(t) = \sum_{i=1}^{n} \left\lceil \frac{t+T_i}{T_i} \right\rceil M_i b$ note that function $g(t) - \alpha t$ can be maximal only for those values of t that correspond to a step of the ceiling term of $g(t)$. The values are of the form $t = kT_i + \epsilon$ with k being a non-negative integer and $\epsilon > 0$ arbitrarily small. Conversely, if $g(t) = \beta^{max} t$, being both the latter function and αt monotonic increasing, function $g(t) - \alpha t$ can be maximal only for those values of t for which at $t' = t + \epsilon$ (when $\alpha \leq \beta^{max}$) or $t' = t - \epsilon$ (when $\alpha > \beta^{max}$), with $\epsilon > 0$ arbitrarily small, it holds $g(t') \neq \beta^{max} t$. These values of t must be an intersection between the two components that define $g(t)$, which are those of the set ψ. Lemma follows. □

Lemma 4 provides a practical test to ensure that no task can find the NI queue full. Furthermore, Δ_{NI}^{max} provides a transmission bound to be used in the timing characterization of the system.

V. Conclusion and future work

In this article, we presented an analysis on the different type of latencies that can be introduced by the communication interface when a task want to send out data packets. Furthermore, we derived a model, an analytic condition and respective formal proofs to ensure that introduced latencies, especially regarding the queueing, are bounded. In the future, we are willing to extend this work including a fault model that takes into account the transmission error and error-recovery strategies also providing a response time analysis model for tasks. Further, we may investigate whether a different task scheduling model (e.g. a sporadic sender task) may introduce lower pessimism in the analysis.

REFERENCES

[1] Muhammad Aamir and Mustafa A Zaidi. A buffer management scheme for packet queues in manet. *Tsinghua Science and Technology*, 18(6):543–553, 2013.

[2] Dimitri Bertsekas and Robert Gallager. *Data Networks (2nd Ed.)*. Prentice-Hall, Inc., USA, 1992.

[3] B. Brandenburg. Scheduling and locking in multiprocessor real-time operating systems. In *Ph.D. dissertation, The University of North Carolina at Chapel Hill*, 2011.

[4] Mike Jia, Jiannong Cao, and Lei Yang. Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing. In *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 352–357, 2014.

[5] J.F. Kurose and K.W. Ross. *Computer Networking: A Top-Down Approach*. Pearson Education, Limited, 2010.

[6] Jinke Ren, Guanding Yu, Yunlong Cai, and Yinghui He. Latency optimization for resource allocation in mobile-edge computation offloading. *IEEE Transactions on Wireless Communications*, 17(8):5506–5519, 2018.

[7] Jinke Ren, Guanding Yu, Yinghui He, and Geoffrey Ye Li. Collaborative cloud and edge computing for latency minimization. *IEEE Transactions on Vehicular Technology*, 68(5):5031–5044, 2019.

[8] Dario Sabella, Alessandro Vaillant, Pekka Kuure, Uwe Rauschenbach, and Fabio Giust. Mobile-edge computing architecture: The role of mec in the internet of things. *IEEE Consumer Electronics Magazine*, 5(4):84–91, 2016.

[9] Lak Sad. Parallelising reception and transmission in queues of secondary users. *International Journal of Electrical and Computer Engineering*, 9(4):3221, 2019.

[10] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. Lavea: Latency-aware video analytics on edge computing platform. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, pages 1–13, 2017.

Safety Verification of Third-Party Hardware Modules via Information Flow Tracking

Andres Meza*, Francesco Restuccia*, Ryan Kastner*, and Jason Oberg†
*University of California San Diego
†Tortuga Logic Inc, San José

***Abstract*—Modern System-on-Chip (SoC) architectures are heterogeneous consisting of hundreds of IP cores and shared on-chip resources. On-chip communication uses simple and efficient standards like AMBA AXI and TileLink. These communication standards focus on performance and are often underspecified with respect to safety and security. If used incorrectly, this opens the door for nefarious behaviors, which are especially dangerous in mission-critical applications that have tight constraints on safety and security. These behaviors are compliant with the on-chip communication standard and therefore can be difficult to capture in a standard verification flow. This paper describes how to use Information Flow Tracking (IFT) to verify the safety of bus interactions among on-chip hardware resources. Our methodology is integrated into a safety verification flow leveraging Tortuga Logic Radix-S IFT tool.**

I. Introduction

Mission-critical systems increasingly rely on system-on-chip (SoC) architectures to deliver high performance and meet real-time constraints. For example, deep neural networks are commonly executed on custom on-chip hardware accelerators that perform object detection, image classification, and other critical computing tasks.

Due to the critical nature of the operating tasks, mission-critical systems have strict safety and security requirements. Among them, *timing predictability* is crucial – the system must be able to correctly operate its critical functionaries within a given *deadline*. Breaking such timing constraints can cause dramatic consequences.

Heterogeneous systems are composed of multiple specialized modules. It is common practice to integrate third-party modules alongside in-house modules to obtain a platform with the desired functionality. Communication among these modules and all of the on-chip resources is facilitated by a system interconnect implementing an on-chip communication protocol. In this paper, we use AMBA AXI due to its widespread usage throughout the industry. A similar analysis is possible with other on-chip communication protocols, which we leave as future work.

Generally, each module, be it a third-party module or an in-house developed module, utilizes the same communication protocol as the system interconnect to communicate with the rest of the system. Thus, the role of the system integrator is to verify that all modules, especially third-party modules, strictly adhere to the requirements set out in the standard for that protocol.

However, even if every module passes the aforementioned verification, this does not mean that the system is free of communication-related vulnerabilities. Previous works have demonstrated how a lack of specification in the on-chip protocol definition can be exploited by hardware modules to generate conditions endangering the execution of the entire system [1]–[3]. Such conditions must be avoided in any mission-critical system.

We explore the use of hardware information flow tracking tools to verify the safety requirements of on-chip communication. Hardware information flow tracking (IFT) is a verification technique that enables the tracking of information as it propagates through the hardware [4]. Hardware IFT techniques have been developed and popularized to identify security vulnerabilities in hardware modules throughout the development lifecycle [5]–[8]. Tortuga Logic Radix-S is a simulation-based IFT tool for security verification [9]. We use Radix-S for our experiments, but note that our techniques generalize to other commercial formal IFT tools.

We demonstrate our safety verification methodology to verify an AXI bus stall issue caused by a lack of specification in the AMBA AXI standard [1]. Our verification is performed on a design using AXI-compliant hardware modules implemented on a Xilinx FPGA multi-core SoC platform. Although we focus on a specific issue of the AMBA AXI standard in this paper, our methodology can be extended for the verification of other weaknesses or vulnerabilities related to AMBA AXI and, eventually, to other popular on-chip communication standards (TileLink, Wishbone).

In the next section, we introduce an example SoC architecture and describe the AXI bus stall problem. Section III describes our safety verification methodology and demonstrates its effectiveness for the AXI bus stall problem. We conclude in Section IV and provide some directions for future work.

II. Motivations and Background

The AXI standard leaves great flexibility in the definition of bus transactions. If not properly managed, such flexibility has been demonstrated to be the source of unpredictable behavior ranging from uneven bandwidth distribution [2] to complete system deadlocks [1]. This section briefly introduces the architecture under analysis and the safety issue under consideration.

A. Sample SoC architecture

A typical System-on-Chip (SoC) architecture is composed of a set of controller devices C (e.g., processors, hardware accelerators, DMAs, etc.) sharing a set of peripheral devices P (e.g., memory controllers, GPIOs, etc.). Controllers and peripherals communicate through a system interconnect. We assume that the system interconnect is based on the AMBA AXI standard [10]. A generic SoC architecture deploying N controllers $(C_1, \ldots, C_N)$ and L peripherals $(P_1, \ldots, P_L)$ is depicted in Figure 1. Each of the controllers $(C_1, \ldots, C_N)$ exports a manager (M) interface. Each of the peripherals exports a subordinate (S) interface. The AXI interconnect I_{AXI} arbitrates the access of the controllers to the peripherals.

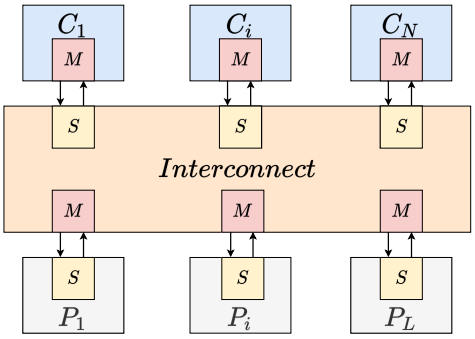

Fig. 1: The sample System-on-chip architecture deploying N controller modules (C) and L peripheral modules (P).

Bus transactions are issued by controllers and served by peripherals. An AXI bus transaction starts with the issue of an address request from the generic controller C_i. The AXI interconnect samples the address request and routes it to the destination peripheral P_j. The address request is served by P_j, which provides the requested read data (in the case of a read transaction) or accepts the provided write data and acknowledges with a write response (in the case of a write transaction).

B. The AXI bus stall problem

Consider a two-controller (C_0 and C_1), one peripheral (P_0) architecture (see Figure 1 with $N = 2$ and $L = 1$). To demonstrate the issue on real hardware, we consider the AXI SmartConnect [11] as the system interconnect, which is the state-of-the-art interconnect for Xilinx systems. AXI SmartConnect implements a round-robin arbitration to solve conflicts among controllers. We consider that each controller issues a single write request. This simple configuration is enough for showcasing the AXI bus stall problem. It is worth mentioning that the same considerations hold for architectures involving more controllers/peripherals and issuing multiple requests. The described system has been deployed on the Xilinx ZYNQ Ultrascale+ SoC platform, where C_0 and C_1 are two custom high-performance DMAs, while P_0 is the shared DRAM memory controller of the platform. The conditions generating the AXI bus stall problem are briefly described next. A full description of the AXI bus stall problem is reported in [1].

(1) Assume that C_0 issues a write request A_0 directed to P_0. Once issued, A_0 is sampled by I_{AXI} and routed to P_0. Similarly, assume that in the same time frame C_1 issues a write request A_1 directed to P_0.
(2) Assume that the round-robin arbitration at I_{AXI} is won by A_0 – this means that A_0 is propagated to P_0 by I_{AXI} before A_1.
(3) According to the AXI standard, after a write request is issued, the controller should provide the corresponding data to be written in the peripheral. However, AXI does not define any time limit for the controllers to provide the data words after a request has been granted.
(4) If C_0 delays the data provisioning for A_0, the service of A_1 is delayed – A_0 has been propagated first by I_{AXI} to reach P_0. This means that even if C_1 is ready to provide the data corresponding to its transaction A_1, such data cannot be propagated by I_{AXI} to P_0 until C_0 completes the provisioning of all of the data words corresponding to A_0 (data interleaving is forbidden in write transactions beginning in AMBA AXI4 [10]).

The delay possibly introduced by C_0 is fully compliant with the AXI standard. According to AXI, C_0 can delay its data provisioning for an unbounded time after booking the bus by issuing the address request. This means that C_0 can leverage this lack of specification provided by the standard to directly affect the availability of the shared peripheral P_0 to the other controllers in the system (in the specific case, C_1). A similar issue can happen for read transactions – the description is omitted for brevity.

III. Safety Verification Methodology

In the realm of security verification, many tools enable system integrators to specify security properties and then check if their system adheres to these properties. Some of these tools rely on formal methods in order to carry out this check while others rely on simulation-based methods. Due to the scaling issues associated with formal methods [4], the safety verification methodology we propose in this section relies on simulation-based tools. We leave the investigation of the trade-offs between formal methods and simulation-based methods for future works. In this section, we introduce a method capable of detecting the AXI bus stall problem described in Section II-B through the use of a commercial simulation-based IFT tool, supported by a custom-developed, parametrizable trigger module.

A. Addressing the AXI bus stall problem

At its core, the AXI bus stall problem described in Section II-B is caused by controllers not being constrained to provision data within a limited amount of time after booking the bus with a write request. In order to mitigate against this, system integrators need to verify that each controller provisions data within a limited amount of time. The acceptable amount of time varies depending on the constraints of the

system. Once an integrator determines how much delay each controller can safely introduce into the system, they can follow the proposed methodology outlined in Section III-C to verify that each controller meets the appropriate delay requirement.

B. The Trigger Module

Our verification in Section III-C relies on a custom, parameterizable module (i.e., the trigger module) to track the state of the write transactions for a single controller. The inputs to this module are a signal specifying the maximum delay limit for the controller and the incoming and outgoing AXI signals of the controller's write-related channels. The output of this module is a signal indicating the state of the controller with respect to write transactions. Specifically, the module outputs whether the controller is in one of three states: (1) idle (i.e., not in a transaction), (2) in a transaction and provisioning data within the delay limit, or (3) in a transaction and provisioning data outside of the delay limit. With this module and the information it provides, system integrators can verify the safety of the controller using the approach described in the following section.

C. Simulation-Based IFT for Safety Verification

Our safety verification approach relies on simulation-based information flow tracking. This approach requires a design, a testbench, IFT properties, and an information flow tracking tool (e.g., Tortuga Logic's Radix-S). The following steps outline the flow of this verification approach when using Tortuga Logic's Radix-S.

1) Determine the Delay Limits: The first step in the safety verification approach determines the appropriate delay limit for every controller C_i in the system. A controller's delay limit is the maximum amount of delay (measured in clock cycles) the controller can safely introduce into the system. System integrators must determine this based on the constraints of their system. In hard real-time systems, this can be achieved by applying the results of worst-case analysis bounding the response time of the hardware modules deployed in the system [12]–[14]. In systems dealing with softer timing constraints, profiling and over-provisioning techniques can be evaluated as an alternative strategy.

2) Insert the Trigger Modules: The second step inserts a set of trigger modules T into the existing design. Since each trigger module T_i can only track the write transaction state of a single controller, system integrators should add a trigger module for every controller they wish to verify. Figure 2 depicts a generic SoC architecture deploying N controllers ($C_1, \ldots, C_N$) and L peripherals ($P_1, \ldots, P_L$) with the addition of K trigger modules ($T_1, \ldots, T_K$) for verification purposes (note that $0 \leq K \leq N$). Given the simplicity of the trigger module's design, we measured a minimal impact on the overall simulation time of the system. This means that system integrators could add a trigger module for every controller in their system (i.e., $K = N$) with minimal overhead.

3) Specify the Safety Properties: In order to verify that a controller meets a certain safety requirement (i.e., it does not introduce a delay larger than its delay limit determined in Step 1), the safety requirement must be represented as a formally specified and evaluable expression containing design signals, explicit values, and operators. We will refer to these expressions as safety properties, and each safety property will be specified as an IFT property. We use IFT properties because they enable us to track a signal as it propagates through a hardware design [4]. The following safety property template, based on the security property templates in [15], specifies that illegally provisioned write data (i.e., data provisioned after a delay limit as indicated by the output of a trigger module T) from some controller C should not flow to the system interconnect I_{AXI}. In other words, this property will track a controller's write data (i.e., `` `C_w_data ``) when that controller's trigger module indicates that it is illegally provisioning data (i.e., `` `T_out `` == 2`b11``), and if any of the illegally provisioned write data flow to the system interconnect (i.e., `` `I_AXI_w_data ``), the property will fail.

```
`C_w_data`            //source sig
  when
  (`T_out` == 2`b11) //tracking condition
  =/=>               //no-flow operator
`I_AXI_w_data`        //destination sig
```

It should be noted that the hardware information flow property above makes use of the no-flow operator (=/=>) in order to indicate non-interference between the source signal and a destination signal [16]. Hardware information flow properties are a type of hyperproperty that are specified over sets of traces and are useful for proving a key aspect of information flow analysis (i.e., non-interference) [15].

4) Generate the IFT Models: The fourth step of the process uses Tortuga Logic's Radix-S tool to generate IFT models for every safety property specified in Step 3. The IFT model generated by Radix-S is a modified version of the design to be simulated that has been instrumented with additional

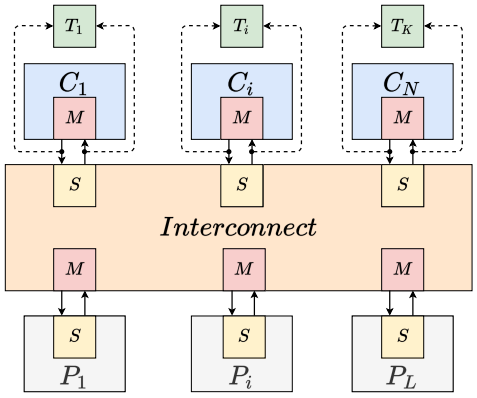

Fig. 2: The sample System-on-chip architecture deploying N controller modules (C) and L peripheral modules (P) with the addition of K trigger modules (T) for verification purposes (note that $0 \leq K \leq N$).

logic in order to enable hardware information flow tracking of the design signals relevant to a particular IFT property. It should be noted that the IFT models generated by Radix-S are typically referred to as "security monitors" or "security models" but, for the sake of clarity, we refer to them as IFT models in this safety verification context.

5) Create Testbench: In the fifth step of the process system integrators create a testbench in order to drive the simulation of the design and IFT models. System integrators are likely to already have a testbench at their disposal for functional verification purposes. Functional testbenches can be reused for this safety verification but they may need to be extended or modified depending on how thoroughly they stimulate the controllers in the system. Interested readers can refer to [17] for methods of determining and increasing testbench coverage.

6) Verify Safety Properties via Simulation: The final step in the safety verification is to verify the specified safety properties via simulation. After the simulation has been completed, system integrators determine which, if any, properties failed and then adequately address the delay introduced by the controllers associated with those failing properties. In the event of a failed property during verification, system integrators should take appropriate countermeasures including but not limited to requesting a module redesign or sourcing alternative modules.

We tested our proposed methodology by using it to identify the AXI bus stall problem in a system integrating fully-compliant AXI modules. To this end, we leveraged the test setup described in Section II-B, modifying the DMA modules to introduce programmable bus stalls during write transactions. As expected, our proposed methodology was able to detect the bus stalls introduced by the DMAs – the safety verification failed any time a DMA module introduced a stall longer than the maximum allowable stalls parametrized in the specific instance of the verification.

IV. Conclusion

We proposed a safety verification methodology utilizing simulation-based information flow tracking for the purpose of verifying the safety of on-chip communication in hardware modules. We validated this methodology by using it to identify fully-compliant AXI controllers which introduced delays capable of causing the AXI bus stall problem via a write transaction.

While the safety verification methodology was focused on addressing the write case of the AXI bus stall problem, there are more safety vulnerabilities allowed for by the AMBA AXI standard that could be identified using a slightly modified version of this methodology. Some of these vulnerabilities include the read version of the AXI bus stall problem fully described in [1], the heterogeneous burst length problem described in [2], and other specific issues that can be generated by behaviors related to transactions IDs, memory protection, and memory buffering. Expanding the framework to consider additional vulnerabilities is a compelling future research direction.

Another interesting direction would explore how other verification techniques could be used to carry out safety verification. For instance, formal methods and standard simulation-based methods (without IFT) could be used to address the safety verification task presented in this paper, albeit with different sets of steps and safety properties. An in-depth analysis could provide valuable insight regarding the trade-offs (e.g., effort required by system integrators, verification time overhead, level of assurance, etc.) between such techniques for safety verification tasks on systems in real scenarios with multiple properties to be verified.

References

[1] F. Restuccia, A. Biondi, M. Marinoni, and G. Buttazzo, "Safely Preventing Unbounded Delays During Bus Transactions in FPGA-based SoC," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2020.

[2] F. Restuccia, M. Pagani, A. Biondi, M. Marinoni, and G. Buttazzo, "Is your bus arbiter really fair? restoring fairness in AXI interconnects for FPGA SoCs," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, p. 51, 2019.

[3] F. Restuccia, A. Biondi, M. Marinoni, G. Cicero, and G. Buttazzo, "Axi hyperconnect: A predictable, hypervisor-level interconnect for hardware accelerators in fpga soc," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[4] W. Hu, A. Ardeshiricham, and R. Kastner, "Hardware information flow tracking," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–39, 2021.

[5] W. Hu, J. Oberg, A. Irturk, M. Tiwari, T. Sherwood, D. Mu, and R. Kastner, "Theoretical fundamentals of gate level information flow tracking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 8, pp. 1128–1140, 2011.

[6] A. Ardeshiricham, W. Hu, J. Marxen, and R. Kastner, "Register transfer level information flow tracking for provably secure hardware design," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1691–1696.

[7] D. Zhang, Y. Wang, G. E. Suh, and A. C. Myers, "A hardware design language for timing-sensitive information-flow security," *Acm Sigplan Notices*, vol. 50, no. 4, pp. 503–516, 2015.

[8] C. Pilato, K. Wu, S. Garg, R. Karri, and F. Regazzoni, "Tainthls: High-level synthesis for dynamic information flow tracking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 798–808, 2018.

[9] *The Tortuga Logic Radix-S offical website*, Tortuga Logic, https://tortugalogic.com/radix-s/.

[10] *AMBA AXI and ACE Protocol Specification*, ARM, 2011.

[11] *SmartConnect, LogiCORE IP Product Guide*, Xilinx, 2018, pG247.

[12] F. Restuccia, M. Pagani, A. Biondi, M. Marinoni, and G. Buttazzo, "Modeling and Analysis of Bus Contention for Hardware Accelerators in FPGA SoCs," in *32st Euromicro Conference on Real-Time Systems (ECRTS 2020)*, 2020.

[13] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo, "A holistic memory contention analysis for parallel real-time tasks under partitioned scheduling," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 239–252.

[14] M. Hassan and R. Pellizzoni, "Bounding DRAM interference in COTS heterogeneous MPSoCs for mixed criticality systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2323–2336, 2018.

[15] F. Restuccia, A. Meza, and R. Kastner, "Aker: A design and verification framework for safe and secure soc access control," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[16] A. Sabelfeld and A. C. Myers, "Language-based information-flow security," *IEEE Journal on selected areas in communications*, vol. 21, no. 1, pp. 5–19, 2003.

[17] S. Tasiran and K. Keutzer, "Coverage metrics for functional validation of hardware designs," *IEEE Design Test of Computers*, vol. 18, no. 4, pp. 36–45, 2001.